



**NAVAL
POSTGRADUATE
SCHOOL**

MONTEREY, CALIFORNIA

THESIS

**IMPLEMENTATION OF A FAULT TOLERANT CONTROL
UNIT WITHIN AN FPGA FOR SPACE APPLICATIONS**

by

Gaspar M. Perez Casanova

December 2006

Thesis Co-Advisors:

Herschel H. Loomis, Jr.
Alan A. Ross

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE December 2006	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE: Implementation of a Fault Tolerant Control Unit within an FPGA for Space Applications		5. FUNDING NUMBERS
6. AUTHOR(S) Gaspar M. Perez Casanova		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		12b. DISTRIBUTION CODE
13. ABSTRACT (maximum 200 words) The space environment implies a challenge for the development and utilization of electronics. Field Programmable Gate Arrays (FPGAs) represent a possible solution to that challenge. An FPGA itself is not a Fault Tolerant component, but with the correct configuration it can emulate and behave as one. The Configurable Fault Tolerant Processor (CFTP) developed at the Naval Postgraduate School (NPS) was intended to work as a platform for the implementation and testing of designs and experiments for space applications. The major components of the CFTP are two FPGAs, one configured as the control FPGA (X1) and the other as the experiment FPGA (X2). The configuration of the experiment FPGA already includes fault tolerant properties against radiation and its effects over FPGAs. The control experiment did not have any fault tolerance built-in.		
This thesis investigates the design, considerations, implementation, performance and resource utilization of a Fault Tolerant Control Unit based on FPGA technology using a Triple Modular Redundancy (TMR) approach.		
14. SUBJECT TERMS Field Programmable Gate Array (FPGA), Triple Modular Redundancy (TMR), Resource Utilization, Single Event Upset (SEU), Optimization, Manual Injection of Errors		15. NUMBER OF PAGES 105
		16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified
		20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)

Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**IMPLEMENTATION OF A FAULT TOLERANT CONTROL UNIT WITHIN AN
FPGA FOR SPACE APPLICATIONS**

Gaspar M. Perez Casanova
Lieutenant, Mexican Navy
B.S., Heroica Escuela Naval Militar, 1998

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
December 2006**

Author: Gaspar M. Perez Casanova

Approved by: Herschel H. Loomis, Jr.
Thesis Co-Advisor

Alan A. Ross
Thesis Co-Advisor

Jeffery B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The space environment presents a challenge for the development and utilization of electronics. Field Programmable Gate Arrays (FPGAs) represent a possible solution to that challenge. An FPGA itself is not a Fault Tolerant component, but with the correct configuration it can emulate and behave as one. The Configurable Fault Tolerant Processor (CFTP) developed at the Naval Postgraduate School (NPS) is intended to work as a platform for the implementation and verification of designs and experiments for space applications. The major components of the CFTP are two FPGAs, one configured as the experiment FPGA and the other as the control FPGA. The configuration of the experiment FPGA already includes fault tolerant properties against radiation and its effects on FPGAs. The control FPGA did not have any fault tolerance built-in.

This thesis investigates the design, considerations, implementation, performance and resource utilization of a Fault Tolerant Control Unit based on FPGA technology using a Triple Modular Redundancy (TMR) approach.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	COMPUTER SYSTEMS FOR SPACE APPLICATIONS.....	2
1.	N-Modular Redundancy.....	3
2.	N-Version Programming (NVP)	4
3.	Error Coding Techniques.....	5
B.	SPACE ENVIRONMENT AND ITS EFFECTS ON ELECTRONICS	5
1.	Space Environment.....	6
a.	<i>Van Allen Belts</i>	6
b.	<i>Cosmic Rays</i>	6
c.	<i>Solar Flares</i>	7
2.	Radiation Effects.....	8
a.	<i>Total Ionizing Dose (TID)</i>	10
b.	<i>Dose Rate Upset and Latchup</i>	10
c.	<i>Displacement Damage</i>	10
d.	<i>Single Event Effects</i>	11
C.	RAD HARDENING OF ELECTRONIC DEVICES.....	12
1.	Radiation-Hardening Techniques	12
a.	<i>Physical Techniques</i>	12
b.	<i>Logical Techniques</i>	13
2.	Performance Implications	14
D.	CFTP PROGRAM.....	14
E.	RESEARCH OBJECTIVES.....	15
F.	RELATED WORK	16
G.	CHAPTER SUMMARY.....	16
II.	FPGA ARCHITECTURE	19
A.	FPGA CLASSIFICATION	19
1.	SRAM-Based FPGAs.....	19
2.	Floating Gate Programming	20
3.	Anti-Fuse Programming.....	21
B.	SRAM-BASED FPGA ARCHITECTURE.....	22
1.	Configurable Logic Blocks (CLBs).....	23
2.	Input/Output Blocks (IOBs).....	24
3.	Block SelectRAM	24
4.	Routing Resources	24
5.	Delay-Locked Loop	24
C.	RADIATION EFFECTS ON FPGAS	24
1.	SRAM-based FPGAs Radiation Effects.....	24
2.	Floating Gate FPGAs Radiation Effects	25
3.	Anti-Fuse FPGAs Radiation Effects.....	26
D.	FPGA COMPARISON.....	27

E. CHAPTER SUMMARY.....	28
III. CONFIGURABLE FAULT TOLERANT PROCESSOR (CFTP) OVERVIEW.....	29
A. CFTP ARCHITECTURE	29
1. Experiment FPGA (X2).....	31
2. Control FPGA (X1).....	32
3. EEPROM.....	32
4. Flash Memory.....	33
5. SDRAM.....	33
6. PC/104 Bus.....	34
B. OPERATION OF THE CFTP BOARD	34
1. Boundary Scan Mode (JTAG)	34
2. SelectMAP Mode.....	35
3. Master Serial Mode.....	35
4. Slave Serial Mode.....	35
C. CHAPTER SUMMARY.....	36
IV. CONTROL FPGA (X1) ARCHITECTURE	37
A. CLOCK GENERATOR (CLOCKGEN.VHD)	38
B. PC/104 INTERFACE (PC104INT.VHD).....	39
C. X2 INTERFACE (X2INT.VHD).....	40
D. SELECTMAP CONFIGURATION (selectmap_config.vhd)	40
E. SELECTMAP READBACK (selectmap_readback.vhd).....	41
F. TOP LEVEL MODULE (top_level.vhd)	41
G. CHAPTER SUMMARY.....	41
V. FAULT TOLERANT CONTROL FPGA (FTX1).....	43
A. FAULT TOLERANT TECHNIQUES APPLICABLE TO FPGAS	43
1. Circuit Level Modification Method.....	44
2. High-Level Description Method	45
B. IMPLEMENTATION OF FAULT TOLERANT CONTROL FPGA (FTX1).....	47
1. 3PFTX1 Implementation.....	48
2. BFTX1 Implementation.....	50
3. MFTX1 Implementation	53
C. CHAPTER SUMMARY.....	55
VI. EVALUATION OF THE FAULT TOLERANT CONTROL FPGA (FTX1).....	57
A. HARDWARE UTILIZATION IN THE FTX1 IMPLEMENTATIONS	58
1. Contol FPGA (X1) Resources	58
2. Fault Tolerant Contol FPGA (BFTX1) Resources	60
3. Fault Tolerant Contol FPGA (MFTX1) Resources	61
B. EVALUATION OF THE FTX1 IMPLEMENTATIONS THROUGH HARDWARE UTILIZATION	62
C. EVALUATION OF FUNCTIONALITY AND PERFORMANCE OF THE FTX1	65
D. MANUAL INJECTION OF ERRORS	69

E. CHAPTER SUMMARY.....	72
VII. CONCLUSIONS AND RECOMMENDATIONS.....	73
A. SUMMARY	73
B. CONCLUSIONS	74
C. FOLLOW-ON RESEARCH.....	75
LIST OF REFERENCES	77
INITIAL DISTRIBUTION LIST	83

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1.	Van Allen Radiation Belts (From Ref. [2])	7
Figure 2.	Heliosphere (From Ref. [3])	8
Figure 3.	Radiation Effects on Electronics (From Ref. [4]).....	9
Figure 4.	Single Event Effects (From Ref. [7]).....	11
Figure 5.	SRAM-based FPGA (From Ref. [14]).....	20
Figure 6.	Floating Gate Programming (From Ref. [15]).....	21
Figure 7.	Antifuse FPGA Concept (From Ref. [14])	21
Figure 8.	SRAM-based FPGA Architecture (From Ref.[16]).....	22
Figure 9.	Configurable Logic Block (From Ref.[16]).....	23
Figure 10.	CFTP Development Board (From Ref. [24]).....	30
Figure 11.	X1 Architecture (From Ref. [24].).....	38
Figure 12.	Triple Modular Redundancy for Xilinx FPGAs (From Ref. [42]).....	45
Figure 13.	Configuration Memory of an FPGA (From Ref. [38])	47
Figure 14.	3PFTX1 Implementation	49
Figure 15.	TMR Implemented with Three Output Voters. (From Ref. [41]).....	51
Figure 16.	BFTX1 Implementation	52
Figure 17.	MFTX1 Implementation	54
Figure 18.	Output from X2 through X1	67
Figure 19.	Output from X2 through BFTX1	68
Figure 20.	Output from X2 through MFTX1	68
Figure 21.	Output from X2 through MFTX1 with Manual Injection of Errors	70
Figure 22.	Manual Injection of Errors in MFTX1.....	71

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Radiation Effects (From Ref. [5]).....	8
Table 2.	Comparison between FPGA Types.....	27
Table 3.	XQVR Family Information (From Ref. [25]).....	31
Table 4.	Distribution of Configuration bits in the XQVR 600 (From Ref. [33]).....	58
Table 5.	X1 Resources by Module.....	59
Table 6.	Synthesis Reports for X1	59
Table 7.	BFTX1 Resources by Module	60
Table 8.	Synthesis Reports for BFTX1	61
Table 9.	MFTX1 Resources by Module.....	62
Table 10.	Synthesis Reports for MFTX1	63
Table 11.	Resources of FTX1 Implementations	64
Table 12.	Resources of FTX1 Implementations with Manual Injection.....	71
Table 13.	Resources of FTX1 Implementations	74

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First, I want to thank God for all the help, the faith, love and strength that I get from you every day. I have to say that I would not be able to accomplish this work, I would not have life at all if it was not for you Alicia, my beloved wife, thanks for your love, your support, your patience. I am sorry for the time I spent in front of my computer instead of holding you, I really missed you these two years. I love you more than anything.

I want to thank so many people that have been indispensable in the accomplishment of this thesis.

I want to thank Prof. Herschel H. Loomis, Jr. for the opportunity to be part of the CFTP program, for your teachings that I will carry with me always but more than anything for your patience because I know it was hard to work with me.

To Prof. Alan Ross for your ideas, your energy, your always right vision of things. I really enjoyed working with you. I am going to miss our conversations.

To Major Gerald W. Caldwell for taking care of me. Maybe you did not realize it but you helped me more than you think. Thanks for always answering my questions, thanks for your experience and friendship.

To all the members of the CFTP program, Ron, Rita, David, Mindy, because in one way or the other you were always there to help. You are the base of the success of this program.

To the NPS teachers because these two years you gave me knowledge, reasons, a passion to learn and desire to teach.

To my friends and especially to the Mexican community, whom more than friends are part of my family, I will always have you in my heart and prayers.

To this great country for the opportunity to be here and learn things that I never even imagined.

This work is dedicated to you grandma. When I left Mexico two years ago, I had the feeling that I would not see you again and well, I have you here with me.

EXECUTIVE SUMMARY

The harsh space environment represents a challenge for the design and implementation of computer systems for space applications since radiation can cause an entire system to fail. The basic requirements for space computers are: low power consumption, minimum volume and mass, without sacrificing overall performance. In addition, space computers must exhibit high reliability and should be able to tolerate different kinds of faults. Mandatory properties of space systems are: Fault tolerance, expandability, on-line recovery, and ease of testing and verification. The FPGAs have proved to be a solution for the implementation of computer systems for space.

The purpose of the CFTP program is to provide a platform for the implementation and verification of designs for space applications. The principal components of the CFTP board are two XQVR600 -4CB Xilinx FPGAs, one programmed as the Control FPGA (X1) and the other as the experiment FPGA (X2).

The objective of this work was to make the Control FPGA of the CFTP fault tolerant. In order to accomplish this objective there are a series of fault tolerant techniques applicable to FPGAs that range from the Radiation Hardening of the FPGA to the N-Modular Redundancy.

Triple Modular Redundancy (TMR) was selected since it is a technique well developed and understood and it is the only way to protect the configuration bits stored in the FPGA against Single Event Upsets (SEUs). Three Fault Tolerant Control Unit (FTX1) designs were proposed: 3PFTX1, BFTX1 and MFTX1. All of these implementations are based on the TMR with some variations.

The 3PFTX1 is the first design proposed. It is based on the complete TMR of the original Control Unit (X1). In theory it is the best way to implement a FTX1. From inputs to outputs all the resources being utilized by X1 are replicated including the configuration bits. This provides overall protection to X1 against SEUs.

The restriction with the implementation of 3PFTX1 is that there are no pins available for the triplication of its input and output signals. This limitation makes it impossible to implement 3PFTX1 given that in order to implement complete TMR three different data paths are required starting from the inputs and with just one pin per signal this is impractical. A change of the CFTP hardware is required for the correct implementation of 3PFTX1.

With this limitation in mind a different FTX1 was proposed. BFTX1 is based on the triplication of instances of X1 components declared in the top level module. Although BFTX1 represents a partial implementation of TMR in the original X1, this design increases the reliability of X1 with no detriment of its functionality and performance.

The problem with this design is that since partial TMR is implemented, there are single points of failure in the design. These points of failure are Input/Output Blocks (IOBs), single majority voters and the processes that take place inside the top level module.

The analysis of the synthesis reports from the Xilinx ISE design tools of BFTX1 shows that some automatic optimization was occurring. The hypothesis was that since the instance of a component is being triplicated (three instances per component) the Xilinx tools are smart enough to detect this fact and optimize the design by deleting or replacing the redundant resources.

To investigate this hypothesis a third FTX1 approach was proposed. MFTX1 is based on the triplication of components before being instantiated inside top level. With this implementation the same single points of failure present in BFTX1 are also present here.

The analysis of the synthesis reports of MFTX1 shown that the amount of resources utilized in BFTX1 and MFTX1 are pretty much the same. This result discarded the hypothesis but other questions emerged. The optimization of resources occurs in both FTX1 implementations nonetheless, it was not a consequence of the triplication of

instances but a consequence of the design of the original components of X1. Therefore, the TMR in BFTX1 and MFTX1 is implemented properly but the components being replicated are the already simplified ones.

The best way to test the FTX1 designs is to expose the CFTP board to radiation but this option was not available at the time of development of this work. Limitations in hardware made impossible the implementation of a means for the injection of errors and scrubbing of FTX1. Instead, manual injection of errors (through the modification of the outputs one of the redundant modules) was used. This demonstrated the correct implementation of the FTX1 designs.

The automatic and uncontrolled optimization produced by the Xilinx tools does not degrade the functionality and performance of the original X1 and the FTX1 implementations.

TMR excels as the Fault Tolerant technique to implement a FTX1 within an FPGA. The hardware limitations could not be fixed since this work was based on the already implemented and functional X1. Modifications to the original hardware of the CFTP board and more specifically, the utilization of an FPGA with a package with more pins could have solved the problem.

An analysis of the amount of resources utilized by the FTX1 designs, the correctness of their functionality and the manual injection of errors were used to determine the feasibility of the use of TMR in an FPGA for the implementation of a Fault Tolerant Control FPGA. Further radiation testing is required in the different FTX1 designs to determine their effective cross section and the increment of reliability.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Radiation effects in space on electronics are inevitable, but military and space applications working in that environment require high reliability at the lowest possible price. One solution to achieve this goal is the use of Application Specific Integrated Circuits (ASIC) but the cost of this solution and the time of production are large. A better approach is the use of Field Programmable Gate Arrays (FPGAs) because of their commercialization, flexibility, field programmability, reliability, and reconfiguration on the fly. Every system depends on a Control Unit that realizes functions like the interfacing with the outer world, configuration and verification of other components, and space systems are not the exception; these systems require that every component be fault tolerant. A fair question to ask at this point is how can FPGAs be used as control units for space applications in such a way that performance, reliability and low cost come together? What should be considered to design and implement a Fault Tolerant Control Unit in an FPGA? What kind of design should it be? How much room do we need in an FPGA to realize the implementation?

Now days there is a deep understanding about the challenges and risks that exist when we take electronics to space. The necessity of expensive and sophisticated satellites for communications, weather, military applications, etc. has pushed the study of space radiation not just to the knowledge of its effects but to finding effective and efficient solutions to the use of electronics in space applications. One solution involves the design and manufacturing of ASICs with the appropriate characteristics to survive in space. This approach implies high investments in the development of electronics that are used for few critical applications, and even this fact does not certify that the equipment is going to survive to that environment. Instead of the use of ASICs, FPGAs have become an effective option to achieve high reliability at low cost; useful properties are derived from its field programmability, reconfiguration on the fly, and commercialization.

The use of an FPGA by itself does not represent a Fault Tolerant device, but it can be configured to act as one. Space systems can be designed in different ways depending on their purpose, but they will always require the implementation of a Control Unit. A

Fault Tolerant Control Unit is required to ensure that even when other components of the system experience errors, the overall system will continue operating and will recover after a reasonable period of time. The area of interest of this work is focused on the design, implementation and testing of a Fault Tolerant Control FPGA (FTX1) for the Configurable Fault Tolerant Processor (CFTP) board using a Triple Modular Redundancy (TMR) approach. This CFTP board consists of: Control Virtex-I FPGA (X1), Experimental Virtex-I FPGA (X2), PC/104 bus for communication and interfacing with the outer world, EEPROM for booting the system, Flash Memory to store the configuration of experiments to be loaded in X2, and SRAM memory to be used by the experiments running in X2.

A. COMPUTER SYSTEMS FOR SPACE APPLICATIONS

Space computer systems impose design constraints that are much more difficult to meet than do commercial systems. A space computer is designed to execute specific tasks with a minimum of supervision. It has to operate properly for long periods of time without human intervention or repair in a harsh environment. Reference [10] mentions the problems with using Commercial Off The Shelf (COTS) parts instead of Application Specific Integrated Circuits (ASICs) in a way that low cost, high functionality and performance can be achieved without sacrificing the high reliability requirements for space applications. In part this is possible due to the advance in technology of commercial parts but since they are not manufactured assuming that they will be used in space their resistance against radiation is poor.

The basic requirements for space computers are: low power consumption, minimum volume and mass without sacrificing overall performance. In addition, space computers must exhibit high reliability and should be able to tolerate different kinds of faults. Mandatory properties of space systems are: fault tolerance, expandability, on-line recovery and ease of testing and verification [1].

Fault tolerance is an important property of space computers since it has an intimate relation with computer reliability. It assures proper operation during critical missions and avoids maintenance during periods of deployment. Space applications require computer systems that provide continuous service in spite of hardware and

software faults. Depending on the system, the fault tolerant techniques used can differ; but in general the most important Fault Tolerant Techniques applicable to space computers are as follows.

1. N-Modular Redundancy

Because of the new technologies developed using semiconductors and its current low cost, the physical replication of hardware has become the most common form of redundancy used in digital systems today. There are three basic forms of hardware redundancy:

Passive redundancy hides the occurrence of faults and prevents the faults from resulting in errors using fault masking. Passive approaches are designed to achieve fault tolerance without requiring any action on the part of the system or an operator.

Active redundancy achieves fault tolerance by detecting the existence of faults and performing some action to remove the faulty hardware from the system. Active hardware redundancy uses fault detection, fault location and fault recovery in an attempt to achieve fault tolerance.

Hybrid redundancy is a combination of active and passive redundancy, using fault masking to prevent erroneous results from being generated and fault detection, fault location and fault recovery to improve fault tolerance by removing faulty hardware and replacing it with spares.

The NMR technique seems to be expensive and in some ways inconvenient but one important thing to notice is that a really reliable system can be implemented just by using a group of not-so-reliable components to be sure that at least a majority will do the work in spite of faults of others. NMR structure uses these replicated components running identical functions and a restoring organ, which restores reliable information at its output from unreliable information at its inputs.

The biggest weakness of NMR is the majority voter. If an error occurs in the voter or the redundancy is exhausted, then the NMR system fails. In order to overcome this difficulty, several methods have been proposed.

Error detectors can be associated with each redundant modular unit, to stop the unit in case that a fault is detected in it.

Built-in self-test (BIST) is employed to enhance testability of the redundant systems during normal operation, but during the testing, the operation of the redundant systems is interrupted.

Concurrent error detection (CED) is used to build a self-checking fault-tolerant system. It consists of a NMR structure and an extra error indicating circuit (EEIC) which monitors the outputs of all the redundant modules and itself to detect errors. The disadvantage of this system is that it is difficult for the EEIC to distinguish whether a redundant module or itself is faulty. The EEIC is similar to the majority voter of a NMR system but has the additional property of self-verification.

2. N-Version Programming (NVP)

It is true that software does not break as hardware does, but instead, software faults are the result of incorrect software design or coding mistakes. The equivalent of NMR in the software scheme is the N-Version programming, and as expected, it consists of N software modules with a voter. One important characteristic of these modules is that each one must be designed and coded by a separate group of programmers. Each group designs the software from the same set of specifications such that each of the N modules performs the same function. Therefore, when a fault occurs, the fault either does not occur in all modules or it occurs differently in each module, so that the results generated by the modules will differ.

The importance of software fault tolerance is easy to see. If we design a microprocessor-based system to be fault tolerant using a NMR configuration, the hardware redundancy is of little use if a single software fault can disable all of the redundant processors.

The NVP presents some defects. First, software designers tend to make similar mistakes. Second, the N versions of a program are still developed from a common specification, so the N-version approach will not allow the detection of specification mistakes. Therefore, there is no guarantee that two completely independent versions of a program will not have identical faults.

Different combinations of the fault tolerance techniques mentioned above can be exploited for the implementation of space systems. A well designed system is resistant to faults, so the probability of having only one fault in the systems is higher than having two or more faults. Reference [32] is a good review of the techniques explained in this section.

3. Error Coding Techniques

Error Coding Techniques are based on the relationship between the function implemented by a system, its inputs and some special property of the outputs that are verified by a checker module. If the output is not the expected one in response to a specific input then the system indicates an error. Depending on the error coding technique employed, the characteristics of the outputs being verified are different, these properties can be the parity of the output, transition count, number of ones and zeros on the signal, etc.

A property of error coding techniques is that they are only capable of detecting one or as much as two errors in the system with limited capability to locate and correct them. The process of correction is carried on by a higher level module. The only contribution of the error coding techniques is the indication of existence of the error and even then some types of errors can pass undetected by error coding techniques.

B. SPACE ENVIRONMENT AND ITS EFFECTS ON ELECTRONICS

Space is a dynamic and harsh environment of great complexity that needs to be well understood in order to design reliable electronics for space applications. Space is not a perfect vacuum; instead, it contains particles charged with energy.

The environment of interest for this work is the near-Earth space, i.e., from the Low Earth Orbit (LEO) to the Geosynchronous Earth Orbit (GEO). The events that occur in this environment are affected by the conditions of the Earth itself and its interaction with the Sun. Some factors that affect this environment include neutral atoms, gases (plasma), energized particles, radiation and electromagnetic fields. These components are not constant and form a part of the dynamism of the near-Earth environment and vary with the position in orbit (e.g., altitude, latitude and longitude), the hour, time of the year, activity of the sun and cosmic rays.

1. Space Environment

Energetic particles that form the ionizing radiation can penetrate the external surface of space devices (spacecraft, satellites). The amount of energy required to penetrate the surface of a device varies with the type of particle, for electrons, this energy is typically above 100keV, while for protons and other ions this is above 1MeV. Neutrons, gamma-rays and X-rays can also be considered energetic particles [33].

The ionizing radiation particles can be divided in three groups depending on its source of radiation (see Table 1):

- Van Allen belts particles
- Cosmic rays.
- Solar flare particles.

a. Van Allen Belts

The Van Allen Belts are a tori of energetic charged particles trapped by the Earth's magnetic field as shown in Figure 1. There are two belts, the inner belt formed by trapped protons (peak intensity at about 3000 kilometers from Earth's surface) and the outer one formed by electrons (peak intensity from about 12,000 to 22,000 kilometers from the surface) with an area of high particle density (South Atlantic Anomaly) due to the displacement of Earth's magnetic field dipoles from the Earth's center. Temporal radiation belts are formed between the inner and outer belts when interplanetary shock waves (cosmic rays and solar flares) hit the magnetosphere [2].

The source of the Van Allen belts particles are the acceleration of lower energy particles by magnetic storm activity and collisions of cosmic rays with atmospheric particles, representing the most important source of energetic particles in the inner belt. The electrons in the outer belt are quite dynamic and their density is much more variable than the protons of the inner belt. The charge and intensity of electrons is affected by solar winds and the interplanetary magnetic fields [6].

b. Cosmic Rays

Cosmic Rays are high energy charged particles coming from outside the solar system that travel at almost the speed of light (Figure 2). Because cosmic rays are electrically charged they suffer deflection by magnetic fields and it is impossible to determine their exact origin and direction. Most cosmic rays are the nuclei of atoms from

the lightest to the heaviest elements in the periodic table. Cosmic rays also include high energy electrons, protons and energetic heavy ions which can deposit significant amounts of energy in sensitive volumes on space devices and cause problems [6].

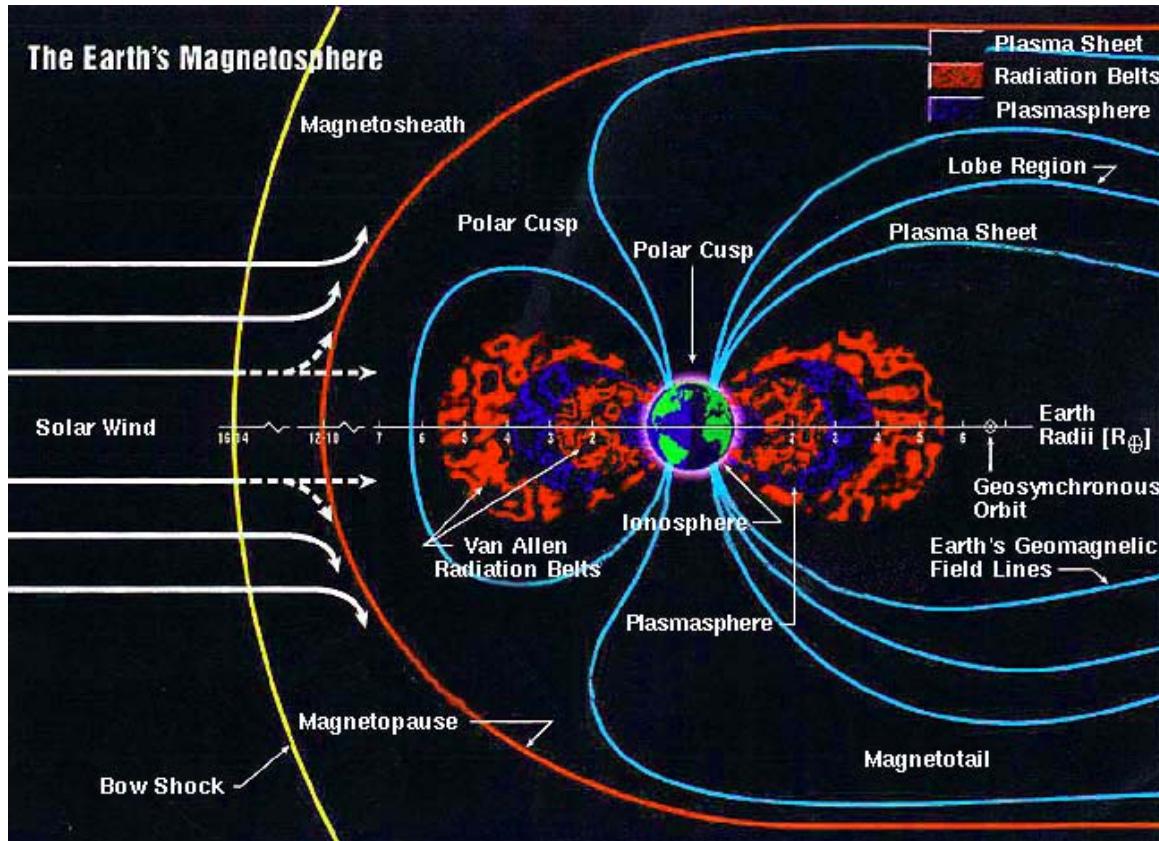


Figure 1. Van Allen Radiation Belts (From Ref. [2])

c. Solar Flares

Solar flares are sudden emissions of optical, UV and X-radiation from an energetic event on the Sun and occur when magnetic energy that has built up in the solar atmosphere is suddenly released. In this complex process, protons and heavy ions are liberated and travel close to the speed of light [6].

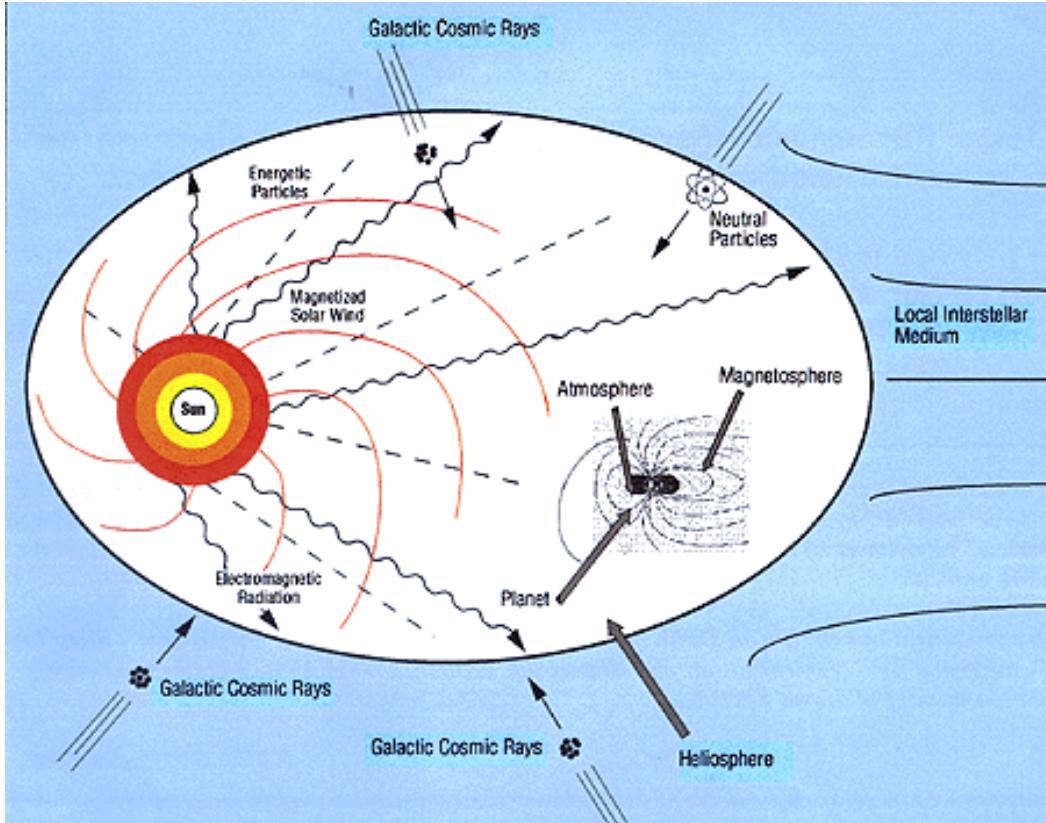


Figure 2. Heliosphere (From Ref. [3])

Radiation Source	Particle Types	Primary Effects in Devices
Trapped radiation belts	Electrons	Ionization damage
	Protons	Ionization damage; SEE in sensitive devices
Galactic cosmic rays	High-energy charged particles	Single-event effects (SEEs)
Solar flares	Electrons	Ionization damage
	Protons	Ionization damage; SEE in sensitive devices
	Lower energy/heavy-charged particles	SEE

Table 1. Radiation Effects (From Ref. [5])

2. Radiation Effects

The key word for electronics for space applications is “radiation.” Radiation is the emission of waves or particles charged with energy. The sources of radiation described above and the particles that launch to space are the principal concerns for the design cycle since each type of particle has specific effects on electronics (Figure 3). The amount of energy absorbed by the device defines the total effects on it. The energy

absorbed turns into electrons and those electrons into charge. The absorption depends on several factors such as the time of exposure to the energized particles, the density of the device being exposed, the mass of the incoming particles, the cross section of the device and the emission rate of particles from the source. The units of absorption of ionizing radiation are Rads.

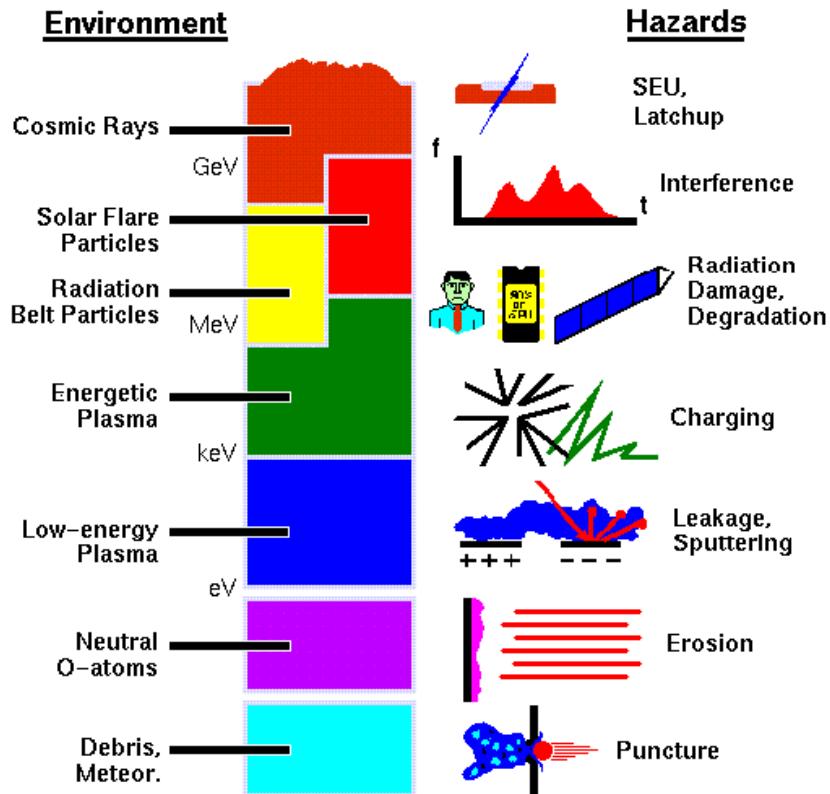


Figure 3. Radiation Effects on Electronics (From Ref. [4])

Essentially radiation can cause two things to electronic devices:

- Remove or ionize electrons from the atoms (Ionizing effects).
- Move or displace atoms in the semiconductors (Displacement Damage).

Ionization effects are caused by charged particles (mostly electrons) and their effects are usually transient, creating glitches and soft errors, but can lead to destruction of the device if they trigger other damage mechanisms like latchup [6].

Displacement damage is caused by neutrons, protons and heavy ions. These particles change the arrangement of the atoms in the lattice causing permanent damage.

More specifically, radiation effects can be categorized in:

- Total Ionizing Dose.
- Dose Rate Upset and Latchup.
- Displacement Damage.
- Single Event Effects.

a. Total Ionizing Dose (TID)

The accumulative radiation over time (TID) is the factor that usually limits the operational lifetime of electronics in space. As the dose accumulates it causes changes on the operational parameters of electronic components outside of their normal range causing the circuit to cease proper functioning. The operational parameters that suffer from total dose are voltage threshold shifts, increased device leakage, power consumption, speed degradation, etc. Total dose greater than 5000 rads delivered to silicon-based devices in seconds to minutes will cause long-term degradation [6]. The total accumulated dose depends on orbit altitude, orientation, and time.

b. Dose Rate Upset and Latchup

When electronics are exposed to high amounts of radiation, ionization occurs over a short period of time (20 to 50 ns) causing large photocurrents [54]. As a result of these photocurrents, circuits may exhibit logic-state upset, and the photocurrents can also trigger destructive latch-up or metal burnout.

c. Displacement Damage

Displacement damage is a cumulative effect caused by prolonged exposure to radiation. It is caused by relatively low-energy atomic particles, as they slow and nearly come to rest. These particles knock lattice atoms out of their locations, creating defects in the crystal structure, which appear as low points in the electrical potential. These points trap conduction electrons, increasing the resistance of the device. Displacement damage is due to cumulative long-term non-ionizing damage from protons of all energies, electrons with energies above 150 KeV and neutrons. Displacement damage degrades minority carrier lifetime; a typical effect would be degradation of gain and leakage current in bipolar transistors [4].

d. Single Event Effects

Single Event Effects, as its name describes, are changes in the parameters of electronic devices due to a single energized particle. Single event effects can be classified in three different categories:

Single Event Upsets (SEUs): Basically this is the change of logic state of an electronic device caused by a particle passing through it. SEUs are transient soft errors, and are non-destructive. Afterward, the device may be re-written into the intended state and continue normal operation. SEUs are more evident in memory modules causing flips on bits of storage data. A deviation of SEUs is multiple-bit SEUs in which a single particle can cause simultaneous errors flipping more than one bit at a time. The worst SEU that can occur is the single-event functional interruption (SEFI) in which a SEU occurs in the device's control circuitry and places the device into an undefined state. The SEFI halts normal operations, and requires a power reset to recover [7].

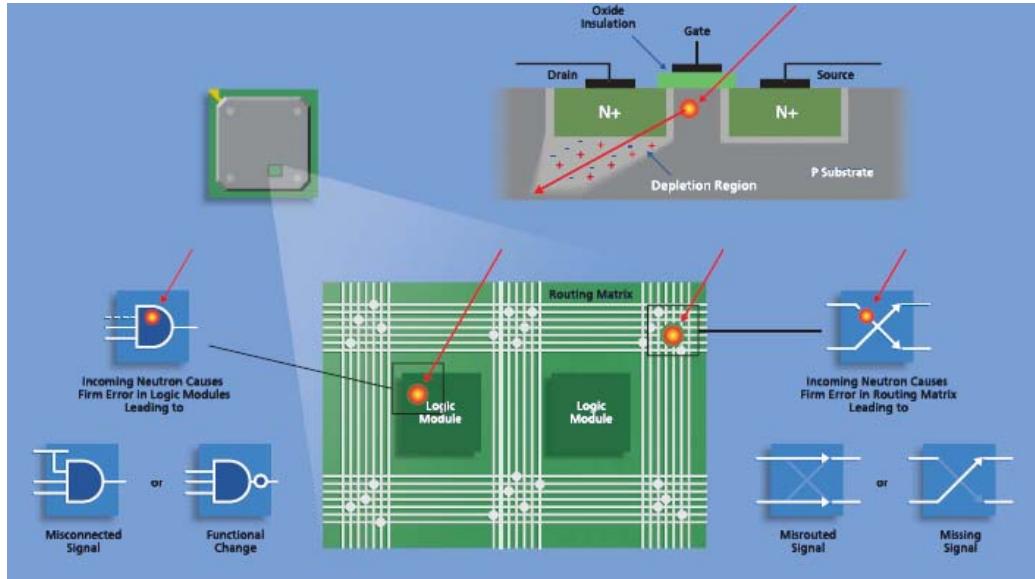


Figure 4. Single Event Effects (From Ref. [7])

Single Event Latchups (SELs): Is a condition in which the device is latched into one logic state and will not change state in response to a logic signal. SELs

are considered hard errors and are potentially destructive. A SEL can be cleared by turning off the device. SELs are strongly temperature dependent: the threshold for latchup decreases at high temperature, and the cross section increases as well [8].

Single Event Burnouts (SEBs): Is a condition in which the current through the device is not limited. SEBs cause the device to fail permanently and its susceptibility has been shown to decrease with increasing temperature [8].

C. RAD HARDENING OF ELECTRONIC DEVICES

Space systems are exposed to radiation than can reduce or end the useful lifetime of their electronic components. In consequence, electronics for space applications must have a means to protect themselves against the effects of radiation. The source of radiation may be energetic particles from the Van Allen belts, galactic cosmic rays or solar flares. In addition to the natural sources of radiation mentioned above, satellites may be exposed to bursts of radiation coming from human produced nuclear explosions.

The amount of radiation that a system needs to be able to resist depends on its location in orbit. At LEO the total radiation dose on an electronic component can be few kilorads, while inside the Van Allen belts the level of radiation can increase to several hundred kilorads [5].

1. Radiation-Hardening Techniques

The objective of radiation hardening techniques is to increase the capability of electronic components to handle specific radiation levels. These techniques can be described in two categories: physical design techniques and circuit design (logic) techniques [6].

a. Physical Techniques

The physical Rad Hard techniques are concerned with the layout of the device. The first approach is the Silicon On Insulator (SOI) technique in which the electronic devices are manufactured on insulating substrates instead of just a silicon wafer. Very thin layers of oxide are used to form the insulator of the devices and regions between the transistors in order to limit the parasitic current. The trade off is the increase on manufacture complexity since thin layers are more difficult to deal with and one imperfection can mess up the entire device. Therefore, the temperature of manufacture is limited by the level of temperature that the thin oxides can resist before loosing its

properties. A similar approach to SOI is the use of Guardbands; these are heavily-doped regions formed by ion implantation that effectively shuts off radiation-induced parasitic leakage paths [54].

The small dimensions of modern electronics make them more susceptible to radiation effects. Therefore, the use of oversized transistors can be implemented in order to improve SEE immunity [54].

If it is not possible to change the manufacturing process, an alternative approach is to modify the package of the device so it can provide a protecting shield against radioation.

The material of the wafer (Normally Silicon) can be changed to one with higher activation energy and bandgap levels, so the energetic particles of radiation do not induce enough energy in the substrate to produce parasitic currents and changes in the electrical properties of transistors. Materials being investigated are silicon carbide and gallium nitride.

Physical Radiation Hardening Techniques are based on changes in the layout of the device resulting in reduction of the performance and increase of the area of the electronic components. The challenge is to find the best results between the radiation hardening of the device and its efficiency and reliability.

b. Logical Techniques

Instead of changing the design of the device at the fabrication level, the radiation hardening can be achieved by changing the properties of the systems where the electronic components are used.

The use of a scrubber circuit can prevent SEE effects by reading periodically the contents of memory and doing verification and correction through Error Detection And Correction (EDAC) algorithms.

A common approach is the use of n-redundant components in hardware and/or software with each one independent of the others. In this way, the computations are executed independently by the different redundant components and the result is compared to determine the correct answer. The disadvantage is the required use of n-

versions of the same component and the logic to vote their outputs, causing an increment in the overall area of the system and the power consumption but with the advantage of self-correction of the system.

A fourth approach is the use of a Watchdog Timer that performs a hard reset of the system if it is not able to verify that the system is working properly.

2. Performance Implications

The challenge on designing radiation hardening electronics is to do it in such a way that the Rad Hard version has a comparable performance with its unhardened counterpart. Since the first parameter to be affected is the overall area of the device, the performance of the Rad Hard version is reduced. A second implication is the power consumption since in the Rad Hard version either bigger transistors or redundant copies of the same circuit are utilized.

D. CFTP PROGRAM

The CFTP program at the Naval Postgraduate School (NPS) was conceived to investigate the applicability of Commercial-off-the-Shelf (COTS) FPGAs in the design and development of reliable computer systems for space applications, to provide reliability, re-programmability, flexibility and low cost to space applications, and the implementation and verification of designs for space applications. Radiation effects in space on electronics are inevitable, but military and space applications working in that environment require high reliability. One solution to achieve this goal is the use of Application Specific Integrated Circuits (ASIC) specifically designed to resist the harsh space environment but the cost and the time of production of this approach are high. An approach that has been investigated is the use of FPGAs that, although they do not have the ability to prevent SEUs and other radiation effects, are able to mask the faults by the means of reconfiguration. This property not only provides a space system with the always required Fault Tolerance capability, it additionally provides flexibility to change the design of the circuit that has been implemented in the FPGA and even to implement an entirely new circuit. The CFTP program looks for the implementation of a reliable platform against radiation effects based on FPGAs that provides to the community a tool to evaluate the reliability and performance of different designs to be used in space applications.

E. RESEARCH OBJECTIVES

The fault tolerance of the CFTP board depends on the reliability of each one of its components. The main components of the CFTP board are two XQVR-4CB228 Xilinx FPGAs, one configured as the Control FPGA (X1) and the other as the Experiment FPGA (X2). The Fault Tolerance of an FPGA depends on the radiation hardening of its resources and the characteristics of the design being implemented on it. Chapter 5 describes the principal Fault Tolerance Techniques applicable to FPGAs.

The principal objectives of this work are the implementation, evaluation and testing of a Fault Tolerant Control FPGA (FTX1) for space applications. Three different designs of a FTX1 were proposed: 3PFTX1, BFTX1 and MFTX1.

The implementation of these designs is based on the Triple Modular Redundancy (TMR) approach with some variations between designs. TMR was selected among the fault tolerant techniques applicable to FPGAs since it is the best way to ensure the redundancy of the configuration memory and user data and the subsequent increment in the reliability of the FPGA. An important point in the implementation of FTX1 is the analysis of the amount of resources utilized by each FTX1 design with the objective to verify that in fact TMR is implemented properly by the Xilinx ISE software and there is not a considerable automatic optimization that can reduce the reliability provided by TMR.

An important requirement for the implementation of a Fault Tolerant Control FPGA is that its functionality stays the same than the original Control Unit without performance loss. The evaluation of FTX1 is carried on through the operation of the CFTP board, the verification of each of its functionalities and the comparison of its output against the original X1 output.

Given that radiation testing was not available at the time of development of this work and that the FPGA used to implement X1 can not be replaced in the short term, manual injection of errors is proposed. This method consists on the modification of one out of three redundant modules so that its output is always in error in comparison with the other two. In this way can be proved that the TMR will detect and correct the intentionally induced errors.

F. RELATED WORK

CFTP is an ongoing program in which several students and academics have participated for several years.

Dan Ebert's Thesis [9] established the initial design of the CFTP architecture. This work includes the analysis of the challenges of electronics for space applications, reviews the different available technologies to implement the basic components of the CFTP board including Random Access Memory (RAM), Read Only Memory (ROM), Flash memory, the kind of FPGAs to be used, interfaces, integration of the components and possible approaches to implement Fault Tolerance of the CFTP.

James Coudeyras's thesis [10], proved by radiation testing the viability of FPGAs for space applications. He implemented a design that used the maximum of the resources inside the FPGA under test to analyze the SEU rate and the efficiency of detection and correction implementations.

Steven A. Johnson's thesis [11] deals with the design and implementation of an entire 16-bit processor with Triple Modular redundancy (TMR) in order to prove the feasibility of the CFTP to implement entire soft-core microprocessors in an efficient and reliable manner.

Peter J. Majewicz's thesis [12] consists of the design and implementation of a five-stage pipelined Reduced Instruction Set Computer (RISC) microprocessor with TMR architecture on an FPGA and the analysis of radiation testing of the TMR system.

G. CHAPTER SUMMARY

This chapter describes the basic requirements of computer systems for space applications, the characteristics of space environment and its effects over electronics, the positives and negatives of Radiation Hardening of electronics, the main objective of the CFTP program and the work done by previous NPS students to make the CFTP possible.

The following chapters describe the classification, architecture and radiation effects on FPGAs, the CFTP board and in specific the architecture of the Control FPGA, the principal fault tolerant techniques applicable to FPGAs and the design implementation and evaluation of a Fault Tolerant Control FPGA.

THIS PAGE INTENTIONALLY LEFT BLANK

II. FPGA ARCHITECTURE

Conventional microprocessors can be programmed to perform a specific task or execute complex computations using basic steps or commands but tend to be slow. ASICs are designed to do a specific computation with high efficiency but are difficult to adapt to new functionalities. FPGAs like ASICs can be programmed to do a specific computation but unlike them, are adaptable (reconfigurable) to new applications. With the use of FPGAs, the functionality of various components can be implemented inside the FPGA saving space and weight without considerable lost of performance.

An FPGA is a semiconductor device that consists of logic components and interconnect that can be programmed in the field after manufacture to execute any function within its logic capabilities. The principle of operation of an FPGA is similar to the one in Programmable Read Only Memory (PROM) or other Programmable Logic Devices (PLDs) with a huge difference in capacity (number of gates) and flexibility of application between these components.

A. FPGA CLASSIFICATION

From the architectural point of view FPGAs are classified in four groups [13]:

- Symmetrical arrays.
- Row-based array.
- Sea-of-gates.
- Hierarchical PLD.

Depending on the programming process used to configure them, FPGAs can also be classified as:

- SRAM based.
- Floating Gate.
- Anti-fuse.

1. SRAM-Based FPGAs

Static Random Access Memory (SRAM) cells are used to store the user's information and configuration data that controls the resources inside the FPGA. Every cell controls a small portion of logic behaving like switches connecting or disconnecting the resources of the FPGA (Figure 5). The advantage of SRAM-based FPGAs is that they

can be re-programmed every time it is necessary and the time to do it is relatively short, increasing their reliability. The main disadvantages are that like any other SRAM component, the information stored in their configuration registers is lost if the power supply goes off requiring an independent booting component to reload the configuration of the FPGA. Another disadvantage is that the SRAM devices also require a relatively large area of silicon because at least five transistors are required for each memory cell of memory[13].

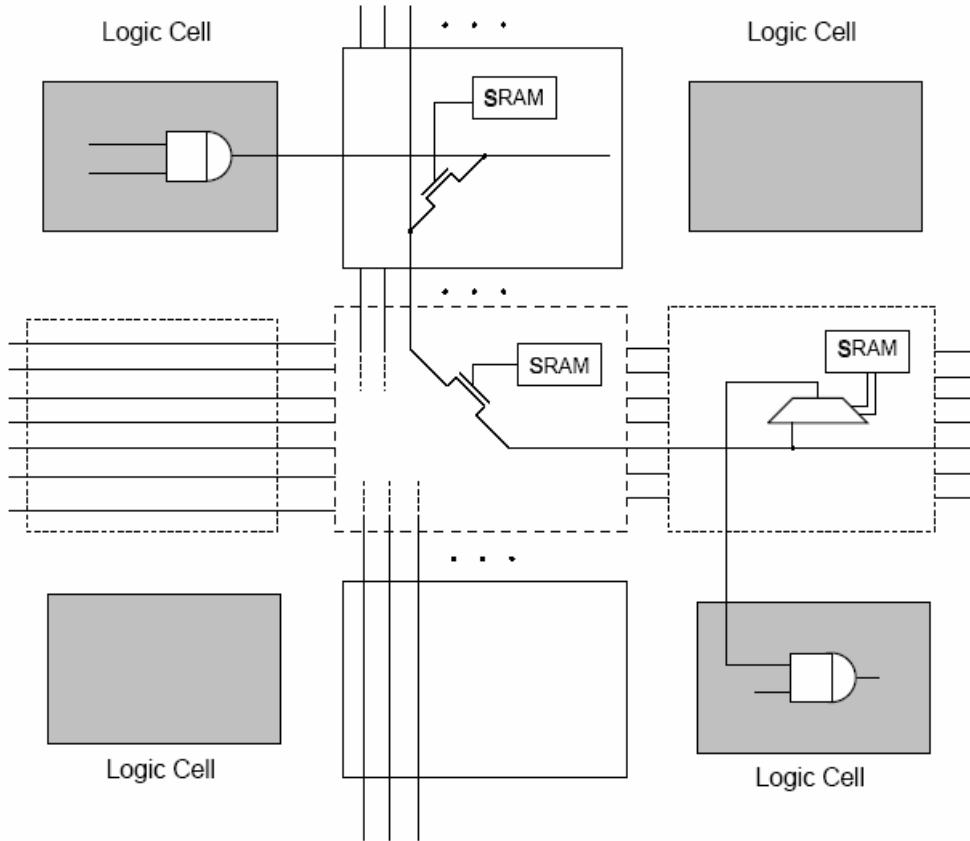


Figure 5. SRAM-based FPGA (From Ref. [14])

2. Floating Gate Programming

In this case FPGAs are based on EEPROM technology (Figure 6). The advantages of this approach are reprogrammability and no loss of information if the power supply goes off. A disadvantage is high static power consumption.

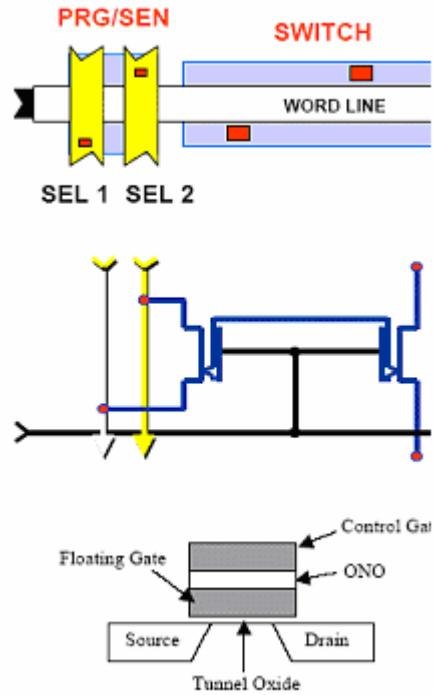


Figure 6. Floating Gate Programming (From Ref. [15])

3. Anti-Fuse Programming

As its name indicates, fuses are used as memory components to store the configuration information (Figure 7). The advantages of anti-fuse FPGAs are that they are relatively small and their tolerance to radiation effects over configuration information. The main disadvantage is that once the FPGA is programmed it cannot be reprogrammed again.

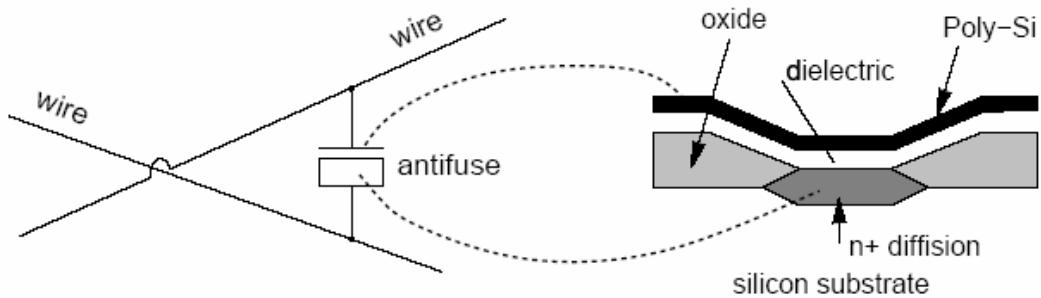


Figure 7. Antifuse FPGA Concept (From Ref. [14])

B. SRAM-BASED FPGA ARCHITECTURE

Every FPGA type has its own specific features. Reference [14] is a good review of the different FPGA architectures.

The architecture of SRAM-based FPGAs is described here since it is the kind of FPGA used in the CFTP board. Xilinx FPGA's basic structure is a symmetric bidimensional array of logic blocks that can be interconnected via horizontal and vertical routing channels (Figure 8). The main components of a SRAM-based FPGA are [16]:

- Configurable Logic Blocks (CLBs).
- Input/Output Blocks (IOBs).
- Block SelectRAM.
- Routing Resources.
- Delay-Locked Loop (DLL).

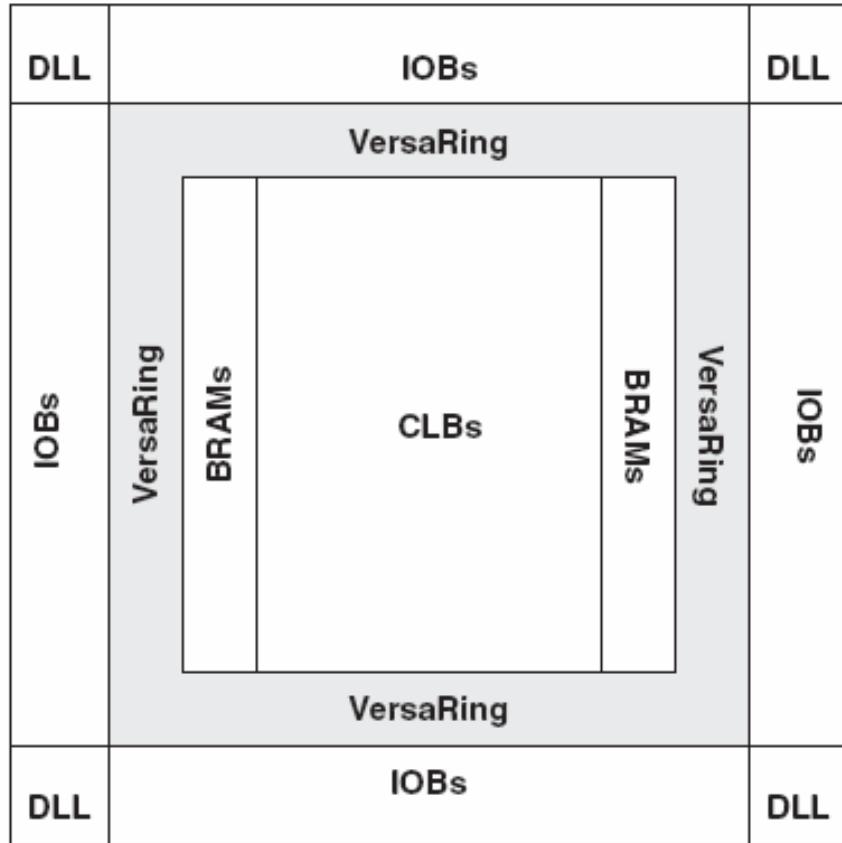


Figure 8. SRAM-based FPGA Architecture (From Ref.[16])

1. Configurable Logic Blocks (CLBs)

The architecture of the entire FPGA is based around the CLBs. These blocks contain all the logic of the FPGA (Figure 9). Each CLB is divided into two slices and each slice contains two logic cells (LCs). Each LC includes a 4-input function generator or Look-up Table (LUT) that can be configured as a 16×1 -bit synchronous RAM and combinations of LUTs provide more complex components like 16×2 -bit or 32×1 -bit synchronous RAM, a 16×1 -bit dual-port synchronous RAM or a 16-bit shift register [16].

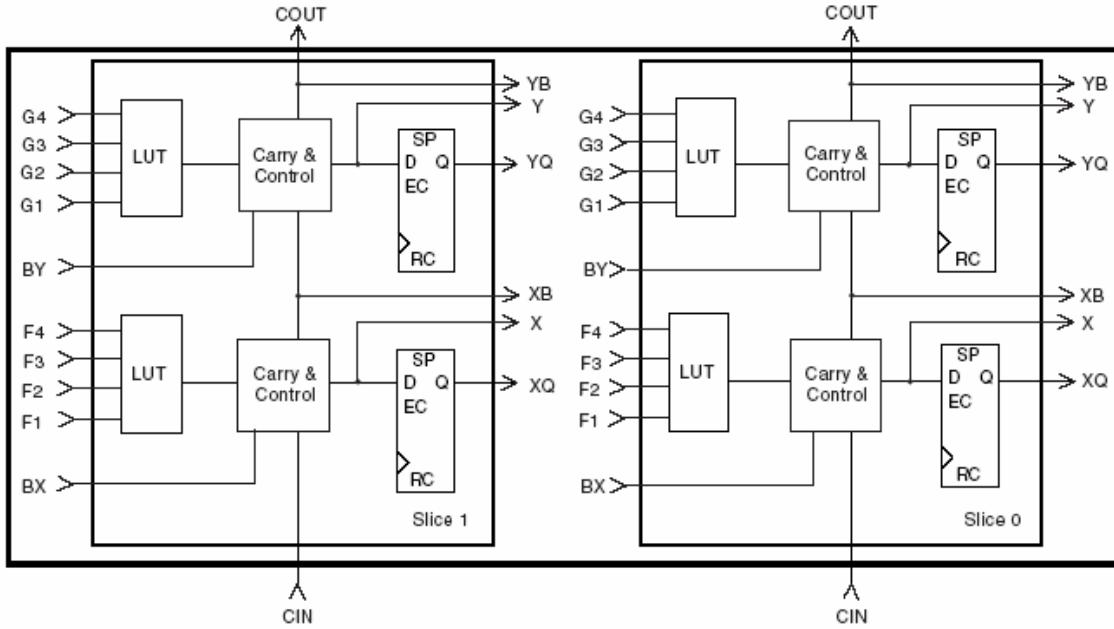


Figure 9. Configurable Logic Block (From Ref.[16])

In addition to the four basic LCs, each CLB contains logic that combines function generators to provide functions of five or six inputs. This additional logic includes flip-flops, multiplexers, arithmetic logic, carry logic, dedicated internal routing and dedicated AND/OR gates.

The configuration data that determines the functions that will be implemented in the CLBs, routing of the outputs and the configuration of the flip-flops are defined in memory cells within the CLB. An alternative mode allows some of these memory cells to be used directly as RAM.

2. Input/Output Blocks (IOBs)

The IOBs can be configured to act as input, output or bidirectional components. Each IOB consists of an input buffer, an output buffer with a three-state, and three storage elements that can be programmed as edge-triggered flip flops or level sensitive latches [16].

3. Block SelectRAM

The Block SelectRAM is additional memory organized in columns along the height of the FPGA. Each Block SelectRAM is a dual-ported RAM with independent control signals for each port.

4. Routing Resources

The Routing Resources consist of an array of routing switches located at the intersections of horizontal and vertical routing channels associated with the rows and columns of CLBs. These resources are configured to connect all the different components of the FPGA as required and also constitute the clock distribution network for the entire FPGA.

5. Delay-Locked Loop

The FPGA contains four dedicated clock pads. Associated with each pad is a Delay-Locked Loop (DLL) that can control skew between the clock input pad and internal clock-input pins throughout the device.

C. RADIATION EFFECTS ON FPGAS

Like any other electronic device, FPGAs suffer from the various effects of radiation but because of their architecture and functions of its components, some effects produce more serious impact than others. And considering that the three FPGAs types mentioned above have different structural components, the principles of configuration and the effects of radiation on them are different.

1. SRAM-based FPGAs Radiation Effects

The most important radiation effects on FPGAs to be considered are the Total Ionizing Dose (TID) and the Single Event Upsets (SEUs).

The operational parameters of semiconductor devices that suffer from TID are voltage threshold shifts, increased device leakage current, power consumption and speed degradation. Several SRAM-based FPGAs have been tested against TID [17, 18] with the

result that above a radiation level of 30 Krad(Si) there is an increment of leakage current and in consequence an increment of power consumption on the chip but no functional failures were found. Another result was that the practical maximum TID that the XQVR Xilinx FPGAs can resist is 100 Krad(Si).

SEUs are of great concern among SRAM-based FPGAs since the configuration data and the user information stored in its internal memory are vulnerable to unintentional change. The susceptibility of SRAM FPGAs is not a secret, different tests have proved [19, 20, 21, 22] the effect of SEUs on FPGAs.

In general, the effects caused by SEUs can be data errors or configuration errors affecting the functionality of the logic modules and the routing between the logic components. Other effects of SEUs are the Single Event Functional Interrupts (SEFIIs) and consist of an alteration of logic content causing a global device function to be activated like a reset and always require a complete reconfiguration for recovery [52]. Reference [18] confirms that the routing resources are the most sensitive to SEU effects since the number of bits devoted to manage interconnections between logic elements and IOBs can be as much as 78% of the total configuration bits.

2. Floating Gate FPGAs Radiation Effects

The basic components of the floating gate FPGA are the flash switch cells that control the configuration of the FPGA and represent the maximum susceptibility of the FPGA to radiation.

The primary effect of total ionizing dose in Floating Gate FPGAs is the propagation delay and standby power supply current [15]. The accumulation of charge in the float gate caused by the ionizing radiation causes the ON or OFF state of the switch to drift closer to the neutral state. For the ON-state the effect is a reduction of the current drive and subsequently increment in the propagation delay. For the OFF-state switch the sub-threshold leakage increases with a subsequent increment in the standby power supply current.

Unlike SRAM switches, FLASH switches are intrinsically hard against SEUs. A SEU occurs when a heavy ion hits a transistor and changes its logic state. In the case of a floating gate the change of charge caused by an ion with LET of 37 MeV-cm²/mg is less than 1% of the total charge on a programmed floating gate [15] and that is not enough to cause a change in the logic state.

An important effect to be considered in Floating Gate FPGAs is the Single Event Latch-up. Supply voltage is the first consideration against latch-up. During programming, the voltage inside a Floating Gate FPGA can be as high as 16 volts instead of the 3.3 volts of normal operation, making the FPGA more vulnerable to latch-up and this must be considered in the design of a system requiring reprogramming of the FPGA.

The Floating Gate FPGA is superior to the SRAM-based FPGA in terms of configuration SEUs tolerance. In contrast SRAM-based FPGAs are superior for in orbit reconfiguration applications.

3. Anti-Fuse FPGAs Radiation Effects

The antifuse FPGA is a device commonly used for space applications for two reasons: its high-density programmable logic and the ionizing radiation tolerance of its programmable switches. In general, anti-fuse FPGAs are considered TID tolerant with one restriction. One of the principal components of the Antifuse FPGA that the (SRAM does not have) is the charge pump circuit. It provides a voltage higher than the supply voltage on the gate of the isolation NMOS transistor so that they can pass VCC signals without degradation. The isolation transistor functions as a turn-off pass gate to isolate high voltage programming voltage (typically $> 2VCC$) from low voltage (VCC) transistors in the logic module [23]. The TID causes the output voltage of the pump to decrease and the supply current increases to a point that can damage the FPGA.

Memory components are susceptible to SEUs but modern Antifuse FPGAs have hardened flip flops making the FPGA more immune to SEUs. Tests performed on Actel Antifuse FPGAs [24] have demonstrated that these FPGAs are SEL tolerant.

D. FPGA COMPARISON

Each type of FPGA explained in this chapter has its specific properties, pros and cons. The factor that determines which one is the best FPGA type is the application for which the FPGA is going to be used.

SRAM-based FPGAs are the largest ones in the market. The susceptibility to TID and Single Event Effects is a factor that would limit its use for space applications. The easy of use, reprogrammability and amount of resources in the chip are important properties that make them candidates for applications that require flexibility.

Floating Gate based FPGAs are less susceptible to TID than SRAM-based FPGAs and are completely fault tolerant to SEUs. The inconvenience in the use of these devices is the equipment and operation required to reprogram these FPGAs.

Antifuse FPGAs are the fastest and densest in the market, the fault tolerance against TID and SEEs have made them one of the devices most used in space applications. The great inconvenience of these FPGAs is that they can not be reprogrammed at all.

Table 2 abstracts the comparison explained in this section.

	SRAM	Antifuse	Floating Gate
Speed	Worst	Best	Medium
Power	Varies	Medium	Best
Density	Worst	Medium	Best
Radiation	Worst	Best	Medium
Reprogrammable	Yes	No	Yes

Table 2. Comparison between FPGA Types

The FPGAs used in the CFTP are the XQVR600 -4CB228; these are Rad-Hard SRAM-base FPGAs. The main reasons for the utilization of this type of FPGA in the CFTP board are the size (amount of resources) and re-programmability on the fly.

E. CHAPTER SUMMARY

This chapter describes the different FPGA technologies and the programming process for each one, the basic architectural components and radiation effects on SRAM-based FPGA with an emphasis on SEUs causing configuration and data errors.

III. CONFIGURABLE FAULT TOLERANT PROCESSOR (CFTP) OVERVIEW

The challenges inherent to electronics for space applications have pushed the development of new technologies and the adaptation of current ones. The first time an FPGA was used for space applications occurred in 1992 in the Aerospace Corporation's Data Processing Unit (DPU) for the SAMPEX spacecraft [53]. This event marked the beginning of a new era in which universal electronic devices have been replaced by FPGAs, with the result of more reliability, less cost and with no substantial loss of performance for space applications

The CFTP program was conceived with the objective to provide the community with a platform in which the implementation and verification of electronic designs for space applications could be carried on both on the ground and in space. In order to achieve this objective, the CFTP needed to be based on COTS components, to have the properties of reliability, reconfigurability, low-power consumption, flexibility and low cost.

The conception of CFTP was not a one person achievement, but the result of the work of academics, technicians and students. Reference [9] explains in detail the reasons for the architecture and selection of the different CFTP components, operation and the expected functionality of the integrated system

A. CFTP ARCHITECTURE

Any computer system contains a Central Processing Unit (CPU), system memory and a storage component. The CFTP consists of a series of COTS components including FPGAs that realize the same functions as in any other computer system but with the additional property that these COTS devices must be able to withstand the harsh space environment, providing, reliability and flexibility to the CFTP.

The use of COTS FPGAs for space applications has increased over time for the only reason that they are good. FPGAs reduce the development time of any application; in fact an FPGA can be launched into space without any specific functionality and later on be configured to execute the desired work or computation. If that initial work has to be

changed at a later time, a new configuration can be loaded into the FPGA providing flexibility. The cost of an FPGA is considerably higher than a microprocessor but its functionality and capabilities are worth it. An FPGA is manufactured like any other electronic component and the risks of failure are pretty much the same but there is a big difference. If a hard fault exists, instead of having to replace the component, the FPGA can be reconfigured again, thus increasing the reliability of the system.

The CTPF board basically consists of 2 Virtex XQVR600 -4CB228 Rad-Hard SRAM-based FPGAs. The first FPGA called Control FPGA or X1 is connected to the space craft Command and Data Handler (C&DH) through a PC-104 bus that acts as the interface between these two units. It is also directly connected to the second FPGA called the Experiment FPGA or X2, which will hold the experiments running on CFTP. Figure 10 illustrates the actual architecture of the CFTP board.

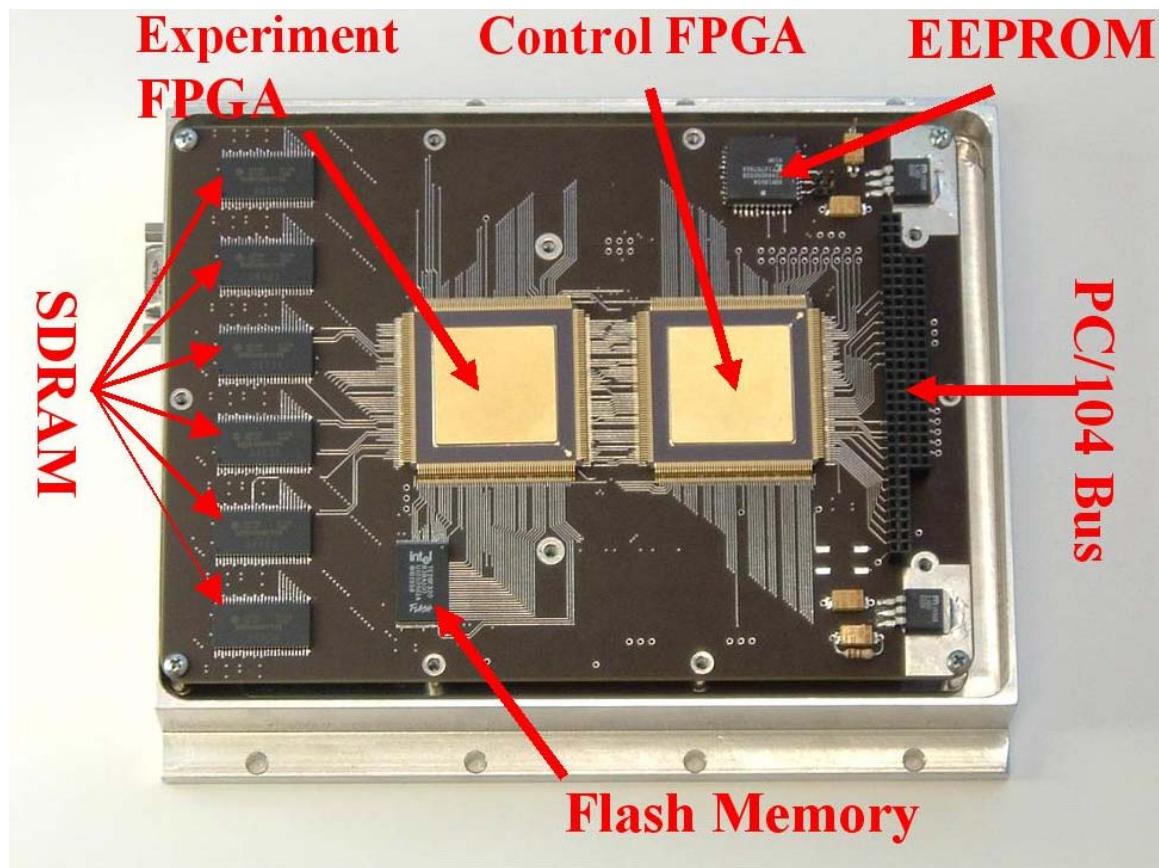


Figure 10. CFTP Development Board (From Ref. [24])

In addition to the 2 Virtex FPGAs, the CFTP contains system memory, storage memory and interfaces to connect the different components inside CFTP and with the outside world.

1. Experiment FPGA (X2)

The central component of the CFTP is the reconfigurable core formed by two FPGAs, the control FPGA and the experiment FPGA. These are Xilinx XQV600 -4C228 FPGAs. Table 3 shows the part number information.

Device	System Gates	CLB Array	Logic Cells	Maximum Available I/O	Block RAM Bits	Max Select RAM Bits
XQVR300	322,970	32x48	6,912	316	65,536	98,304
XQVR600	661,111	48x72	15,552	316	98,304	221,184
XQVR1000	1,124,022	64x96	27,648	404	131,072	393,216

Table 3. XQVR Family Information (From Ref. [25])

The architecture of SRAM-based FPGAs was explained previously in Chapter II. The resources on the FPGA (LUTs, SelectRAMs, Flip Flops, IOBs, etc) are spread across the chip and grouped together in configurable blocks. The basic unit is the CLB. The XQV 600 FPGA has a grid of CLBs distributed in 48 rows and 72 columns with a total of 15,552 logic gates which, when configured, provide functionality to the design. In addition to the CLBs, there are IOBs (Input/Output Blocks), IOIs (IOB Interconnects), 98,304 bits of BRAMs (Block RAMs), BRAMIs (BRAM Interconnects), 221,184 bits of Select RAM and 4 Global Clock (GCLK) blocks.

The X2 is the component of the CFTP where experiments take place which is the main objective of the CFTP program. An FPGA is not fault tolerant by itself meaning that the reliability of X2 will depend in great part on the reliability of the circuit or design being implemented. Therefore, the designers will have the opportunity to create their own designs and expose them to the harsh space environment or a simulation of it and verify that the designs work as desired.

The obvious limitation of X2 is the amount of logic in the FPGA and needs to be considered for the design of applications. Currently, several experiments [11,12,26,] have been implemented successfully in X2. These experiments include a 16-bit Reduced Instruction Set Computer (RISC) microprocessor, a Cordic algorithm, a multiplier, etc.

Even though experimentation is the reason to be of CFTP, X2 is not the only component in CFTP board, a means for evaluating the experiment is required and that is one of the functions that X1 needs to accomplish.

2. Control FPGA (X1)

The control FPGA like X2 is an XQVR600 FPGA. Reference [9] explains that a possibility for the implementation of X1 was to use an anti-fuse FPGA because of its radiation hardness but its limitations in capacity and reprogrammability leaded to the use of SRAM-based FPGAs.

The integration of the different CFTP components takes place in X1. It includes the interface between the control FPGA and the experiment FPGA for verification (Scrubbing) and data flow. It also includes the interface with the PC/104 bus for the data flow between the CFTP and the Command and Data Handler (C&DH) that includes the load of experiment's configuration to the flash memory in the CFTP and the reading of the output of the experiment running in X2.

The functionality of X1 includes:

- Control of X2 configuration and subsequent reconfigurations.
- Verification of the experiment running in X2 through scrubbing.
- Interfacing with system memory (Flash and EEPROM).
- Handling communication with the outside world trough the PC104 bus.

X1 consists of six modules that together provide the functionality explained above. The objective of this work was to make the configuration of X1 fault tolerant and because of the importance of these modules to this thesis, the next chapter is dedicated to explain their functionality.

3. EEPROM

Since the FPGAs used in CFTP are SRAM-based, SEEs represent a problem because data and even worse configuration information can be altered with the

consequent lost of functionality. When a hard fault that X1 can not repair by itself occurs, then the entire CFTP must be rebooted and since one of the characteristics of SRAM-based FPGAs is that it is an entirely volatile device a means to store the initial X1 configuration is required. The device used for this purpose is the Xilinx QPro XQR17V16 EEPROM. It is a One-Time Programmable (OTP) read-only memory designed to store configuration bitstreams of Xilinx FPGA devices with a capacity of 16 Mbits and a TID of 50 Krad [9].

The default mode of operation of the CFTP in flight will involve X1 booting from the Xilinx EEPROM in master serial mode on power-up or reconfiguration. X1 enters the Master Serial mode whenever all three of its select mode pins are low ($M0=0$, $M1=0$, $M2=0$). Data is read from the EEPROM sequentially on a single data line. Synchronization is provided by the rising edge of the temporary signal CCLK, which is generated during configuration. Master Serial Mode provides a simple configuration interface. Only a serial data line, two control lines, and a clock line are required to configure the FPGA. Data from the PROM is read sequentially, accessed via the internal address and bit counters which are incremented on every valid rising edge of CCLK [27].

4. Flash Memory

The CFTP board also contains an Intel TE28F320C3BA 32Mbit flash memory that stores the experiment's configuration to be loaded in X2. This flash memory device provides high-performance asynchronous reads in package compatible densities with a 16 bit data bus. Individually-erasable memory blocks are optimally sized for code and data storage [28]. There are physical connections between the flash and both FPGAs in a way that X1 can read and write to the flash and X2 can read from the flash under the control of X1. This flash memory is capable of holding as many as eight configurations of X2 [9].

5. SDRAM

By definition any computer system has to have system memory and that was the basic reason to include SDRAM in the CFTP board. SDRAM does not participate in the functionality of the CFTP but it is possible that one of the experiments running in X2

requires it. 128 MB of SDRAM Hitachi HM5225405B-75 are included in the CFTP board. This lot of Hitachi SDRAM performed in testing to better than 40 krad TID with an SEL threshold of 46.5 MeV-cm²/mg [29].

6. PC/104 Bus

The PC104 bus is an embedded computer standard controlled by the PC/104 Consortium. The PC/104 bus was designed for applications that depend on reliable data acquisition despite an often extreme environment. The PC/104 bus utilizes 104 pins. These pins include all the normal lines used in the ISA bus, with additional ground pins added to ensure bus integrity [30]

The CFTP has a PC/104 bus between the CFTP board itself and a processor board. This connection provides all communication and file transfer between the CFTP and the spacecraft's C&DH

B. OPERATION OF THE CFTP BOARD

All the processes occurring inside the CFTP are regulated by the Control FPGA and their objective is to configure and verify the experiments running in X2. In order to understand the way CFTP operates, it is necessary to understand the mechanisms through which the FPGAs can be configured. There are four methods available to configure the Virtex FPGAs [31]:

- Boundary Scan mode (JTAG)
- SelectMAP mode.
- Master Serial mode.
- Slave Serial mode.

1. Boundary Scan Mode (JTAG)

JTAG or Boundary Scan mode is an industry standard (IEEE 1149.1, or 1532) serial programming mode. External logic from a cable, microprocessor, or other device is used to drive the JTAG specific pins. The data in this mode is loaded at one bit per assertion of TCK. The Boundary Scan mode is always active when the device is powered [31]. The JTAG mode facilitates test and development of configurations by its design, although it can also be used as a readback and reconfiguration method. The boundary-scan mode is selected by a <101> or <001> on the mode pins (M2, M1, M0).

2. SelectMAP Mode

The SelectMAP mode is the fastest configuration option allowing parallel reading and writing through byte-wide ports. An external clock source, microprocessor, download cable or other FPGA is required in order to provide a byte stream, CCLK, a Chip Select (CS) signal and a Write enable signal (WRITE). The data is loaded one byte per CCLK in this mode. Data can also be read using the SelectMAP mode and multiple Virtex devices can be chained in parallel. SelectMAP provides an “8-bit bidirectional data bus interface” [49], or parallel load capability, for Virtex devices. This mode may be used for both configuration and for readback, and provides a means for the device to be partially reconfigured while it is operating. In this mode, devices may be connected in a parallel-chain, but not serially [31]. After configuration, the pins of the SelectMAP port can be used as additional user I/O. Alternatively, the port can be retained to permit high-speed 8-bit readback. This mode is typically used as a configuration mode when configuration speed is an important factor. The SelectMAP mode is selected by a $<110>$ or $<010>$ inputs on the mode pins (M2, M1, M0).

3. Master Serial Mode

It is the simplest configuration method for FPGAs. The FPGA loads configuration data from a serial PROM. Using the FPGA to provide the clock, it virtually loads itself and utilizes its internal oscillator, which drives the configuration clock. The FPGA provides all the control logic. In this mode, data is loaded at one bit per CCLK. The interface is identical to slave-serial except that an internal oscillator is used to generate the configuration clock (CCLK). Master-serial mode is selected by a $<000>$ or $<100>$ inputs on the mode pins (M2, M1, M0).

4. Slave Serial Mode

In this mode, the FPGA receives configuration data in bit-serial form from a serial PROM or other source of serial configuration data that is loaded at one bit per CCLK. Multiple FPGAs can be daisy-chained for configuration from a single source. Slave serial mode is selected by applying $<111>$ or $<011>$ to the mode pins (M2, M1, M0).

The operation of the CFTP is centered in the process of configuring the experiment FPGA and reading back its configuration, the results of the computations and other data flow.

At power-up the control FPGA boots from the EEPROM in master serial mode. The EEPROM contains the initial configuration of the control FPGA that has the functionality for programming the experiment FPGA from the Flash, communicating with the processor through the PC/104 bus, and reading back the experiment FPGAs configuration and comparing it with the configuration stored in the Flash. Once X2 is in operation, the flow of information between X1 and X2 is carried on through 43 general I/O pins. If it is required, X2 can make use of the SDRAM available in the CFTP board.

X1 has the capability of readback and reconfigure X2 while it is in operation. In order to do it, the FPGA must be set in either JTAG or SelectMAP mode.

C. CHAPTER SUMMARY

This chapter describes the architecture of the CFTP board, the characteristics of its different components and the functions that they execute in the system. A general explanation of the methods to configure FPGAs and the process of operation of the CFTP is also explained.

IV. CONTROL FPGA (X1) ARCHITECTURE

The core of the functionality of the CFTP is the control FPGA (X1). It includes the interface between X1 and X2 for verification (Scrubbing) and data flow. It also includes an interface with the PC/104 bus for the data exchange between the CFTP and the C&DH used to load the experiment's configuration to the flash memory in the CFTP board and to read the output of the experiment running in X2.

The functionality of X1 includes:

- Control of the initial configuration and subsequent reconfigurations of X2.
- Verification of the experiment running in X2 through scrubbing.
- Interfacing with system memory (Flash and EEPROM).
- Handling communication with the outside world through a PC/104 bus.

X1 consists of six Very High Speed Integrated Circuit Hardware Description Language (VHDL) modules that together provide the functionality explained above:

- `clockGen.vhd`
- `pc104IntArm.vhd`
- `selectmap_config.vhd`
- `selectmap_readback.vhd`
- `top_level.vhd`
- `x2Int.vhd`

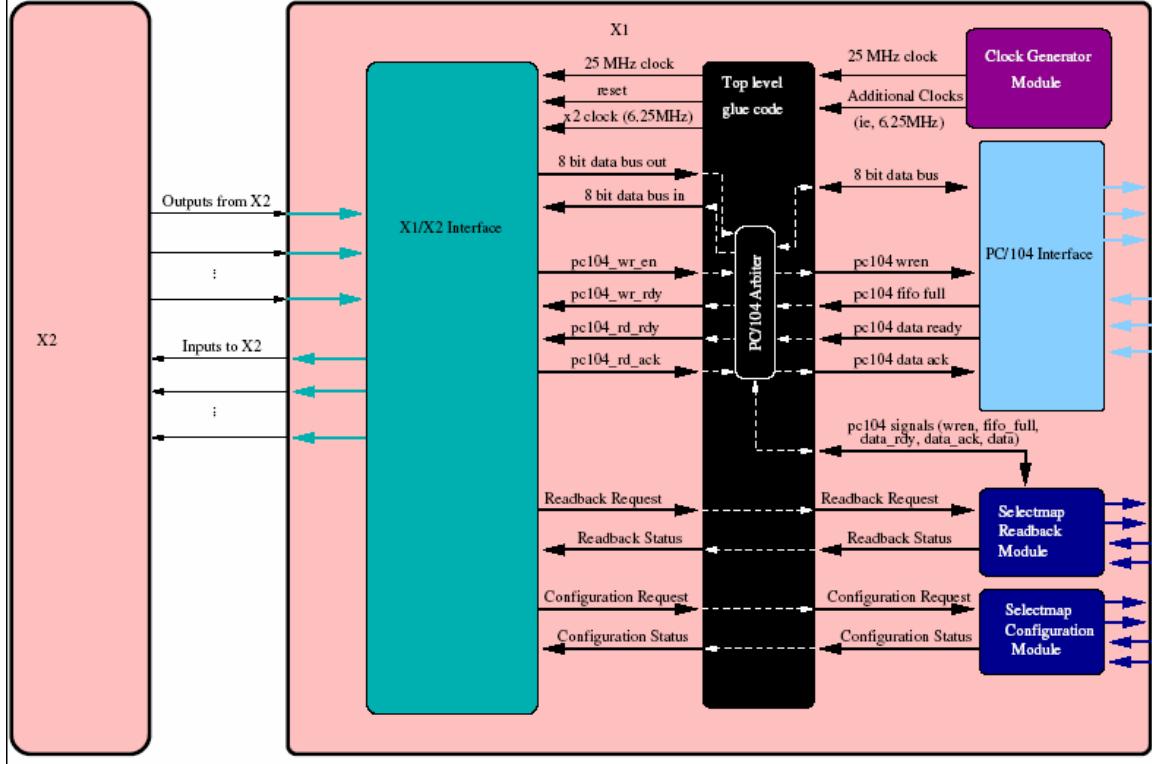


Figure 11. X1 Architecture (From Ref. [24].)

The architecture of X1 is illustrated in Figure 11. Each of the modules accomplishes a specific mission. In the next sections, each module will be explained in detail without any specific order.

A. CLOCK GENERATOR (CLOCKGEN.VHD)

The most important signal for any synchronous component is a clock signal. Even though the CFTP board cannot be considered an entirely synchronous system, synchronization between X1 and X2 is critical.

This module is not exactly a clock generator. The ARM processor connected to the CFTP contains an oscillator that generates a global clock signal at 51 MHz. The X1 components and possibly the experiment running in X2 require a clock signal of different frequency and this is exactly the function of the clockGen.vhd module. Currently the only two output signals of the clock generator module are `s_clock` at 25.5 MHz and `s_clock_X2` at 3.25 MHz. The processes running inside the top level glue of X1 use the

25.5 Mhz clock signal. Depending on the specific design, experiments running in X2 may run at a different speed. The code of the clock generator can be easily modified to get the required clock frequencies.

B. PC/104 INTERFACE (PC104INT.VHD)

The only connection between the CFTP board and the C&DH board is the PC/104 Bus. It is responsible for all data transfers in and out the CFTP to the C&DH. The PC/104 interface module is asynchronous therefore handshaking signals are necessary for the flow control of information passing through it. The internal side of the interface has module ports for write, read and status flags. The status flags display the current state of the FIFO. The FIFO is 32-bits wide and 63-words deep and is generated using the CoreGen tool of the Xilinx ISE v6.1 software.

Basically the pc104Int.vhd module does two functions:

- Control of the flow of information from the experiment FPGA to the ARM processor.
- Control of the flow of information from the ARM processor to the flash memory.

The first step in order to accomplish the functions above is to verify the memory range used by the PC/104 bus, the second step consists of the detection of a reset state and from there the module has to determine if a read or a write needs to be executed.

The read process (Information from the CFTP to the PC/104 bus) starts with a check on a reset signal coming from the ARM side, if not a reset, the process verifies if the next operation is a read from the ARM processor to the FIFO. The read process is controlled by a status signal that indicates when there is information available in the FIFO buffer to be served.

The write process (Information from the ARM to the CFTP) is similar to the read process with a small difference. The write process is executed byte by byte with the control of a status signal that indicates when the FPGA is busy processing a byte and when is ready to receive the next one.

The FIFO buffer uses an interrupt routine to control the flow of data getting in and out of the CFTP. Once the FIFO is full it stops accepting data.

C. X2 INTERFACE (X2INT.VHD)

One of the functions of X1 is the verification (scrubbing) of X2 and the transfer of information between X2 and the ARM processor, in order to be able to do this, X1 needs an interface to communicate with X2 and this is the objective of the x2Int.vhd module.

The main functions of the X2 interface are:

- Reset X2 and the X2 interface signals to specific values.
- Determine the amount of data that has been transferred from X2 to the PC/104 bus.
- Determine the transfer rate of data from X2 to the PC/104 bus.
- Control the rate at which SelectMAP readbacks occur.
- Control the SelectMAP reconfiguration.

The interaction between X1 and X2 must be synchronized and it has to start from a known point at a specific instant. This goal is met by a reset operation. The signal used for this purpose in X1 is RESET_i that sets all x2Int.vhd signals to specific values and the same job is done in X2 by the DATA_TO_X2_RESET_o signal. With this process the operations and communication between the two FPGAs is guaranteed to occur synchronously. The byte size is controlled by a constant value (REPORT_OUT_LENGTH) inside the X2 interface code.

The transfer rate depends on the data size and the value of the sampling rate that is specified by another constant (ERR_RPT_TIME) inside the X2 interface. These values can be easily changed to adapt the interface to the requirements of the experiments running in X2.

The period of time between SelectMAP readback operations is specified by the value of the constant DLY_TIME in the X2 interface code.

The SelectMap reconfiguration is controlled by the number of errors reported by the experiment FPGA. This number is specified in x2Int.vhd code with the err_cnt constant.

D. SELECTMAP CONFIGURATION (selectmap_config.vhd)

The primary function of this module is to program the experiment FPGA. The configuration of X2 takes place following two different events. First X2 is programmed

every time an experiment needs to be loaded in X2 and second, every time the number of accumulated errors occurring in X2 is equal to a threshold value specified by the designer in the x2Int.vhd module. In each of the two cases the SelectMap configuration module reads the configuration data stored in the flash memory starting from address zero and loads the next 900 Kbytes of configuration data into X2. The number of errors that the experiment FPGA can handle depends on the requirement of each experiment and the desires of the designer and there is no rule on the amount of errors that forces the necessity of a reconfiguration of X2.

E. SELECTMAP READBACK (selectmap_readback.vhd)

This module reads all the data stored in the configuration memory of X2 and compares it against the configuration data stored in the flash memory. It is used to verify that the current configuration of X2 is correct providing some level of reliability to the CFTP.

In order to execute a readback or reconfiguration operation specific commands and procedures need to be executed. The entire process is explained in detail in Ref. [31].

F. TOP LEVEL MODULE (top_level.vhd)

This is the module in which all the control FPGA components come together. The basic functions of top_level.vhd are to arbitrate between the other X1 modules by assigning values to the signals that control the processes inside those modules. Additionally top level sets values to signals that are constant like setting the mode pins of X2 to the SelectMap mode.

G. CHAPTER SUMMARY

In this chapter, a description of the architecture of the control FPGA and the functionality and interaction between its six modules has been explained. The following chapter describes the fault tolerant techniques suitable for FPGAs and the three different designs of the Fault Tolerant Control Unit.

THIS PAGE INTENTIONALLY LEFT BLANK

V. FAULT TOLERANT CONTROL FPGA (FTX1)

The objective of the CFTP program is to provide a platform for the verification and testing of designs for space applications. This requires that all its components be fault tolerant. Chapter III describes the tolerance to radiation that every component of the CFTP has embedded on it. The XQVR600 FPGAs are the critical components of the CFTP. They are latch-up immune to a Linear Energy Transfer (LET) of 125MeV-cm²/mg and radiation tolerant to 100krad TID [33] but are quite susceptible to SEUs. Various experiments tested to date in X2 [10, 12] have proved that the X2 design can be fault tolerant but no work had been done in X1 to make it Fault Tolerant.

This chapter describes several fault tolerant techniques applicable to FPGAs and three different approaches to make X1 fault tolerant. These designs are “3PFTX1,” “BFTX1” and “MFTX1.”

A. FAULT TOLERANT TECHNIQUES APPLICABLE TO FPGAS

SEUs are the primary concern of FPGAs for space applications. The effects caused by SEUs in SRAM-based FPGAs depend on the place or resource that the radiation particle hits in the FPGA and the energy level of the particle. The effects of SEUs on FPGAs can be classified into transient errors and permanent errors [46].

Transient errors affect the user-defined logic and flip-flops of the FPGA. The user logic includes block RAM (BRAM), configuration logic block flip-flops and I/O block flip-flops (IOB-FF) [33].

SEUs can cause permanent errors on an FPGA if they alter the contents of configuration memory. Permanent errors are classified into routing errors, LUT bit-flips, and control/clocking bit-flips. Permanent errors are more common than transient errors because the number of SRAM cells dominates the number of user-defined memory elements. Typically, the number of SRAM configuration cells is more than 98% of all memory elements inside an FPGA [46].

The growth of more demanding applications, the advance of the technology and the subsequent vulnerability of smaller and faster components have pushed the development of fault tolerant techniques to increase the reliability of computer systems. Fault tolerant techniques applicable to FPGAs can be classified in two groups [39]:

- Circuit level modification method.
- High-level description method.

1. Circuit Level Modification Method

The circuit level modification method consists in the re-manufacturing of the FPGA by including fault-tolerant elements. This approach may include the replacement of the old and less reliable elements by radiation hardened ones without modifications of the original architecture or the implementation of a new FPGA architecture that improves the reliability of the device. An obvious example of these techniques is the radiation hardening of an FPGA by using larger size components, different manufacturing technologies like Silicon On Insulator (SOI) or the triplication of the resources inside the FPGA [38].

A second approach consists on the use of spare components in such a way that when a fault is detected and located in one of the components of the FPGA, the faulty component is discarded and replaced by another one that was not in use [39]. The spare parts can be distributed in different ways. In Reference [37] two possible distributions of spare allocation are proposed (king-shifting and horse-allocation).

Another option includes the implementation of Error Detection And Correction (EDAC) codes like the Hamming and the Reed-Solomon codes inside the architecture of the FPGA [38], with the limitation that these codes are able to identify just one upset and their implementation for the detection of multiple upsets increases the area and delay overhead of the design [38].

Circuit Level Modification techniques imply a change in the manufacturing process of the FPGA and in consequence an increment in the production cost and the time to market of the product.

2. High-Level Description Method

This method is not based on the modification of the original resources of the FPGA, instead, the design being programmed in the FPGA must include some fault tolerant characteristic. It is normally based in some kind of redundancy so that, if one component fails, the circuit can continue operating with the fault free components [38]. The most common method of fault tolerant redundancy is the Triple Module Redundancy (TMR) with voting [40, 41, 42] first proposed by Von Neumann [51].

The correct implementation of TMR with the Virtex FPGA depends on various factors such as the size and the type of the module to be mitigated. TMR can be implemented based on the module size in four ways [5]: Module Redundancy, Logic Partitioning Redundancy, Logic Duplication Redundancy and Device Redundancy [50]. TMR can also be implemented based on the type of data structure to be mitigated. The logic may be grouped into four different structure types: Throughput Logic, State-machine Logic, I/O Logic, and Special Features (SelectRAM block, DLLs, etc.) [38]. A good reference of the implementation of TMR on Virtex FPGAs is [42]. An example of the implementation of TMR is illustrated in Figure 12.

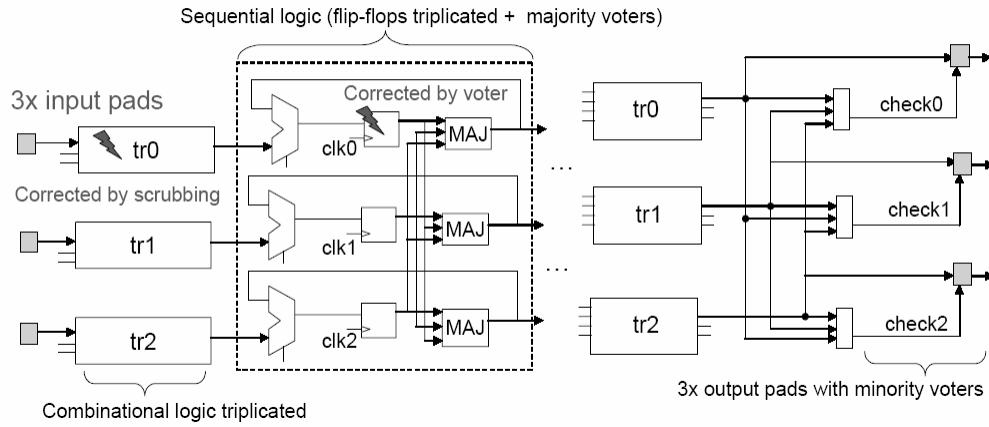


Figure 12. Triple Modular Redundancy for Xilinx FPGAs (From Ref. [42])

Conventional fault-tolerant schemes can only protect user-bits but not configuration bits. The only applicable fault-tolerant mechanism to protect configuration bits is to use Triple Modular Redundancy (TMR) scheme in all used logic and routing

resources [46]. TMR is an attractive solution for SRAM based FPGAs even that the area of the design and amount of resources utilized in the FPGA is increased substantially [33].

The high-level SEU mitigation technique used nowadays to protect designs implemented in the Virtex architecture is mostly based on TMR combined with scrubbing [38]. Scrubbing refers to the periodic readback of the FPGA's configuration memory, comparing it to a known good copy, and writing back any corrections required [49]. Scrubbing allows a system to repair SEUs in the configuration memory without disrupting its operations [44].

TMR is a solid and attractive solution to increase the reliability of FPGAs but it still present some disadvantages [43, 44, 48, 49]:

- Excessive area overhead. The hardened design has 200% more area than the original circuit.
- TMR system can withstand only single upsets at any instant of time, thus, if two redundant modules are simultaneously upset, then the output cannot be guaranteed to be correct.
- The redundant system is considered SEU tolerant under the assumption that the voter circuit is completely immune to SEUs.
- Speed degradation.
- The number of I/O pads available for designers is reduced by three, because the inputs and outputs of each TMR redundant block are triplicated too.
- The power consumption is increased by three as all input and output pins as well as the combinational and sequential logic are triplicated.
- Traditional TMR does not provide a way of re-synchronizing state logic after configuration scrubbing. After a SEU in a traditional TMR state machine is corrected through scrubbing, the state machines must be reset to resynchronize.

This approach does not require re-design of the FPGA or changes in the manufacturing process. Consequently, the cost of implementation is lower than the circuit level modification methods and the designer has the advantage to choose the level of redundancy in his design and the overheads in terms of area, performance and power

dissipation [38]. The fact that TMR does not require changes at the mask level is not exactly a perfect option and it presents some limitations to the designer since there are things inside the FPGA that he can not change, control or even know about.

B. IMPLEMENTATION OF FAULT TOLERANT CONTROL FPGA (FTX1)

The considerations in order to make a device fault tolerant are the device itself, the tasks to be executed by the device, the environment of operation and the requirements of reliability of the system. The architecture and functionality of X1 were discussed previously in Chapter IV. From that information as a starting point, the Fault Tolerant Control Unit was implemented in three different ways for a couple of reasons. First, the fact that the fault tolerant control unit was not designed from scratch, instead, the already functional X1 was modified to include the fault tolerant characteristics represented a limitation on the techniques available. Second, there are in general two methods to implement a circuit in VHDL, architectural and behavioral and the modules and interfaces already existent in the CFTP program were developed with the behavioral method.

The XQVR600 FPGA used to implement X1 has a total of 3,364,928 configuration latches and its configuration bitstream is 3.608 Mbits [25]. All these configuration bits are potentially sensitive to SEUs and consequently, they must be protected by a fault tolerant technique. Figure 12 illustrates the location of configuration memory cells in an FPGA.

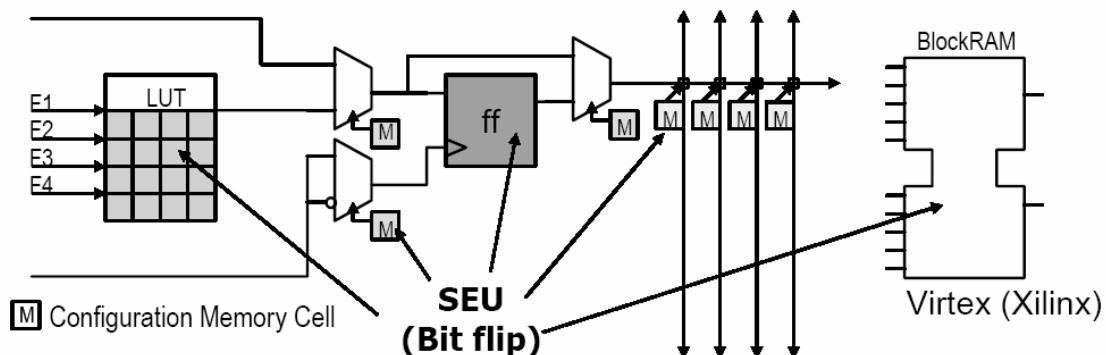


Figure 13. Configuration Memory of an FPGA (From Ref. [38])

The effect of an upset in the FPGA matrix is unique for each type of programmable structure, such as LUT cells, routing cells, flip-flop cells and embedded memory cells. The effects of SEUs in different components of FPGAs have been investigated previously [47, 48].

The TMR technique does not have any rules about which resources of a design to triplicate. The redundancy can go from the triplication of an entire chip (three FPGAs executing the same function instead of one) to the triplication of single logic gates. The level of granularity really depends on the characteristics of the design been triplicated and the requirements of the designer. Every component triplicated becomes fault tolerant to SEUs but at the same time the area of the design increases by three. In consequence, the probability that an energized particle will cause a SEU increases. Even that TMR improves the reliability of a system, and the level of reliability increases with the triplication of every single component, its disadvantages already discussed in the previous section have worst effects, mainly an increment in the cross section of the device [6].

1. 3PFTX1 Implementation

All the functionality of X1 is based on the six modules already discussed in Chapter IV. In order to assure reliability through TMR, the effects of an upset should be isolated inside a single redundant module, so it can not affect the other two replicas of the module.

The first obvious design decision was that all six modules of X1 had to be triplicated and their outputs voted. Where to implement the triplication modules and up to what extent of granularity were the second and not an easy decision to make. As mentioned before, this work did not start from zero; instead, the already functional X1 was modified to include TMR.

From the basic definition of TMR, the 3PFTX1 implementation was based on the triplication of the overall X1. What it means by overall is that the inputs, outputs, components, processes, data paths, arbitration, etc. are triplicated so that indeed the

control FPGA contains three X1s (Figure 14). The granularity of the triplication is at the module level. With this design the hope is that all the components utilized to implement the original X1 will be triplicated without omissions.

The top_level.vhd module contains the entity cftp_arm, which is where the rest of the components are declared and instantiated and the arbitration processes take place. This approach includes the triplication of the top level module and everything included in it.

The problem with the implementation of this design was the FPGA itself. In order to implement overall TMR one basic requirement is that every input and output signal needs three pins in order to be triplicated and the XQVR600-4CB228 FPGA contains just 228 pins already in use. Just three pins are available at this time and the requirement is 121 pins per X1 module plus additional pins for power and ground.

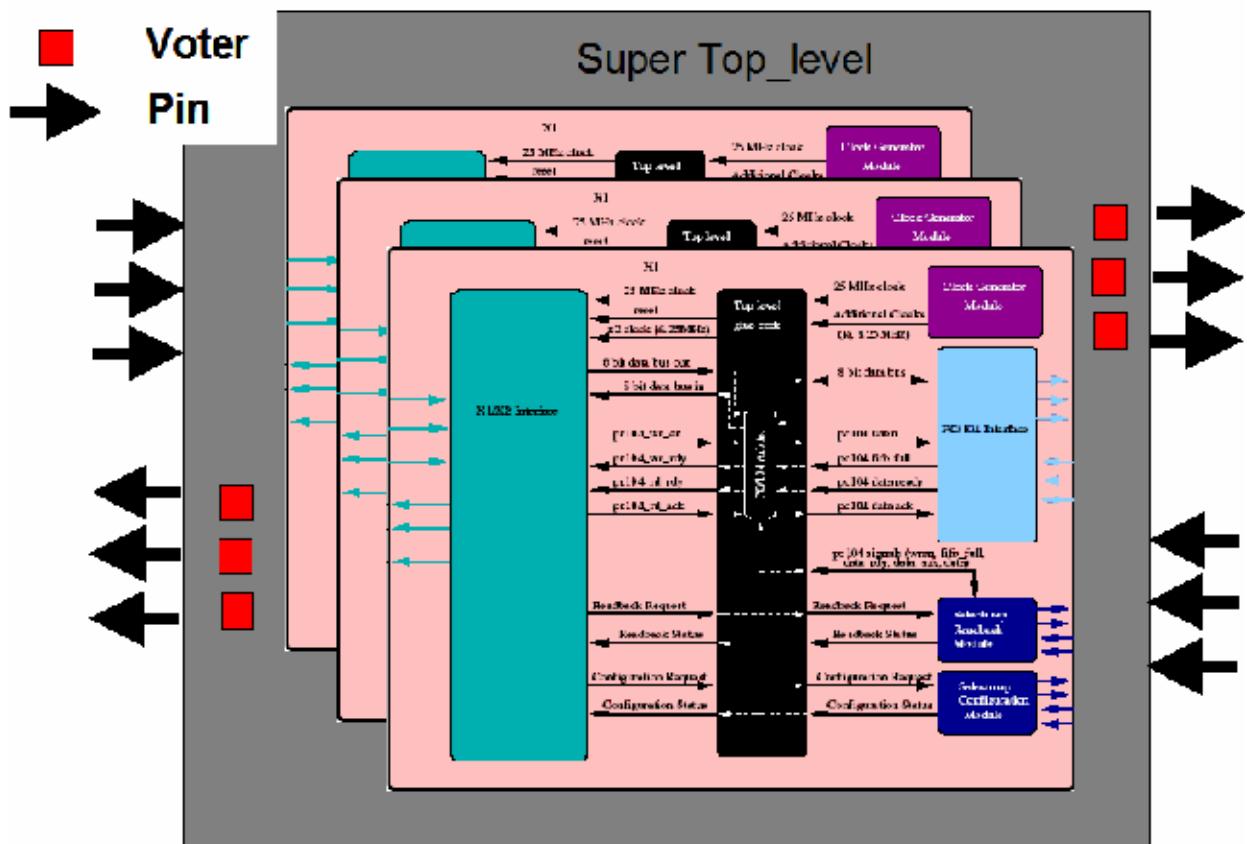


Figure 14. 3PFTX1 Implementation

A second limitation is that a User Constraint File (UCF) is required to define the physical implementation of X1 inside the FPGA and more specifically, the pin assignments. Since the entire X1 is triplicated, 3PFTX1 has 3 different top_level.vhd modules and just one UCF file can be used to define the pin assignments. To solve this limitation a higher hierarchical module was created (Super_top_level). The first option to implement the UCF file for the super top level module was to create a UCF file containing the definition of the pins for the triplicated modules but there are not enough pins in the FPGA, the second option was to triplicate the UCF file and assign one per top_level.vhd module but just one UCF file can be defined per design. The bottom line is that 3PFTX1 was not a viable implementation since three pins per signal are required and the XQVR600 does not have enough pins for this purpose.

Given that the implementation of this design was not possible because of hardware restrictions, another design was proposed. BFTX1 is a modified 3PFTX1 in which all modules (except top_level.vhd) and the functionality of X1 are triplicated. The differences between these two implementations are that in BFTX1 the inputs of X1 come from one pin and then are split into triplicated data flows and that the outputs coming from triplicated modules are voted by a majority voter and then outputted in just one pin.

2. BFTX1 Implementation

The basic idea in the implementation of BFTX1 is to exploit the instantiation of all the X1 components inside the top level module. A very simple method for implementing SEU mitigation in X1 is to triplicate the instances of each one of the X1 components defined in top_level.vhd and vote the final outputs coming from the modules [41]. In order to avoid the inconvenience of having a single point of failure with the implementation of one voter per output signal, three voters per signal were used (Figure 15).

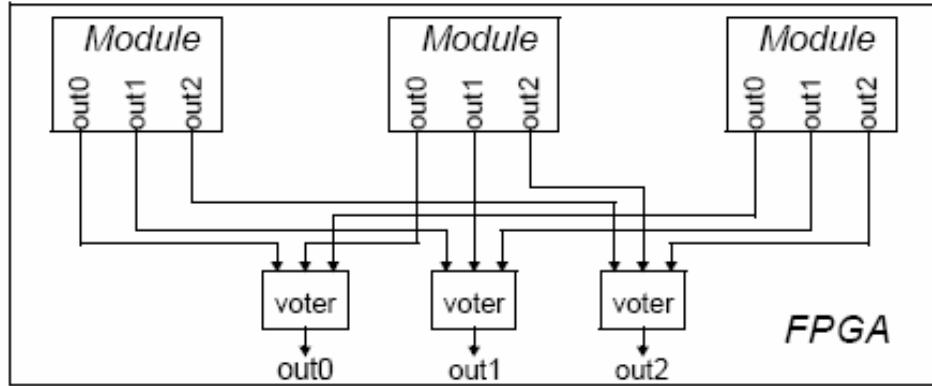


Figure 15. TMR Implemented with Three Output Voters. (From Ref. [41])

The inconvenience of this implementation is that since `top_level.vhd` could not be triplicated for the reasons already explained, just one copy of every signal appearing in this module is allowed, so just one voter per signal was implemented.

Figure 16 shows the implementation of TMR based on the redundancy of instantiations inside `top_level.vhd`.

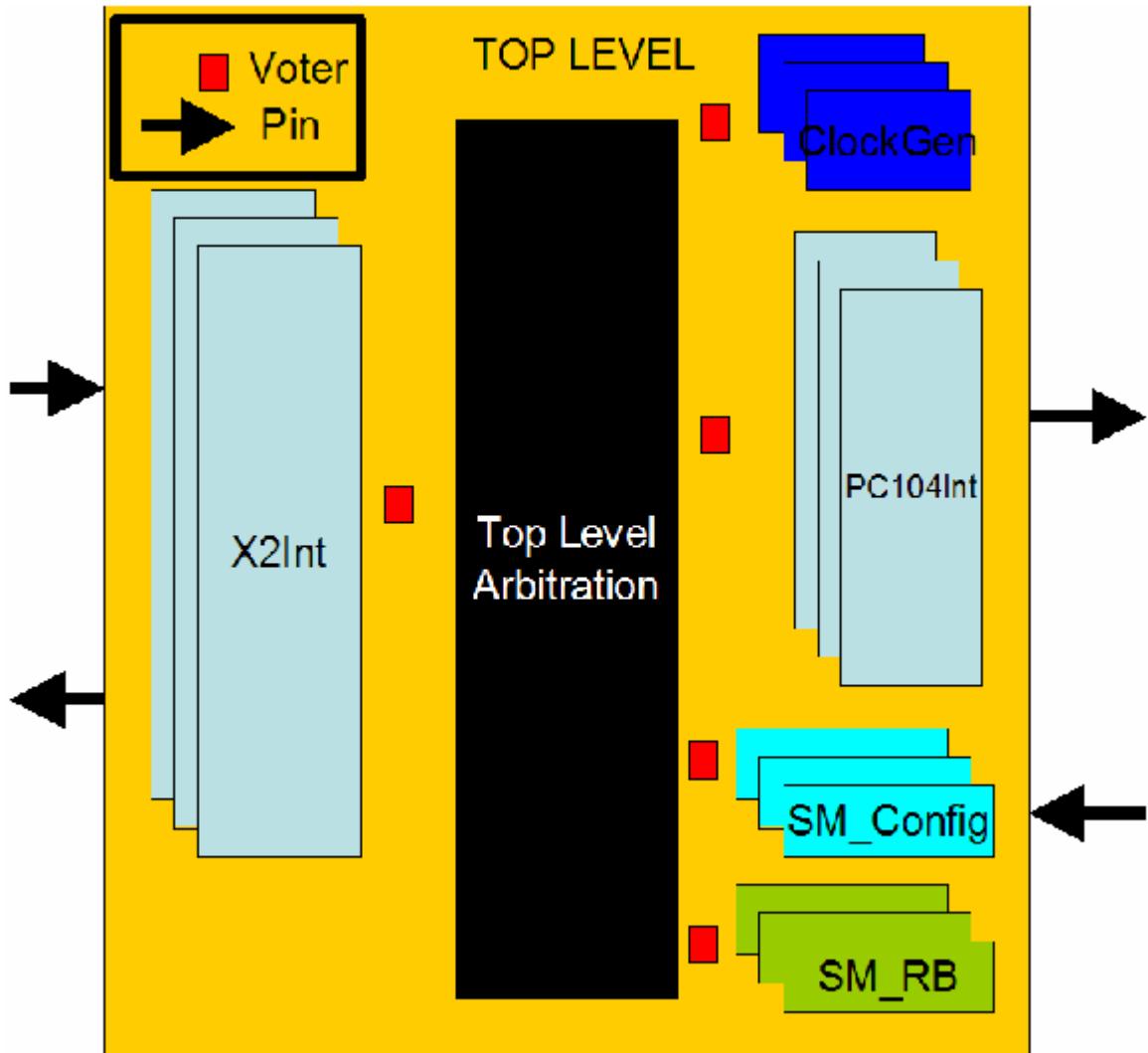


Figure 16. BFTX1 Implementation

The first consideration in this implementation is that any of the input signals of X1 is triplicated, then from a single input point a signal is redirected to three different instances and the same happens with the output signals from X1, in this case the outputs from the triplicated modules go to a final voter and then to a single output pin. Each input and output pin is connected inside the FPGA to a single IOB and since there is not triplication of the signals IOBs represent single points of failure in the same way as one voter per signal.

Another consideration is that the Xilinx ISE software is smart enough to distinguish when a designer is making the mistake of including replicas of a component or a signal and it simplifies the design to achieve the highest performance. The problem is

that the replication of instances is not always a mistake. The BFX1 design uses three instances of the same component. The redundancy of these instances is a primary requirement for the reliability of the design. A theory is that the optimization tool of Xilinx ISE identifies three instances of the same component and automatically simplifies the design by replacing some resources used by others or even deleting redundant components. This optimization is part of the synthesis process of a design and can not be avoided.

The disadvantage of the TMR implementation with redundant instances is that this technique does not provide a simple and robust recovery mechanism after an error has been detected in one of the modules. The errors occurring in any place of an instantiation will not be detected until the output of the module. With combinational logic this is not a problem because the next clock after the detection of the error will load the proper value of the signal in error, but if one of the instantiations contains sequential elements its internal state can be very much different at the output of the redundant modules. The consequence is that the instantiation that presented the error will be useless since the recovery to the proper state of the sequential logic is not automatic. The only method to solve this is to apply some means to resynchronize the modules before another copy of the instantiation presents an error [46].

The advantage of this approach is that the implementation is pretty simple. The only modification that had to be done was the addition of redundant instances in the top_level.vhd module and voters for each one of the output signals.

3. MFTX1 Implementation

Some questions emerged about the inevitable and undesired level of optimization that the Xilinx ISE software implemented in the BFTX1. In order to verify that the Xilinx ISE software used in this work was not simplifying the TMR of X1 to the point that it was non-existent, a different design was proposed in which instead of instantiating the components of X1, they are triplicated as differently named components and then instantiated expecting that this would push Xilinx ISE software to believe that these are indeed different components and to reduce the simplification.

MFTX1 is based as in the previous implementation, on the TMR of the original X1 modules except top_level.vhd for the reasons already exposed. But there is a big difference between BFTX1 and MFTX1. In this case, the instances defined inside top_level.vhd are not triplicated, instead, three copies of the same module are generated as if they were different components and then they are instantiated inside top_level.vhd (Figure 17).

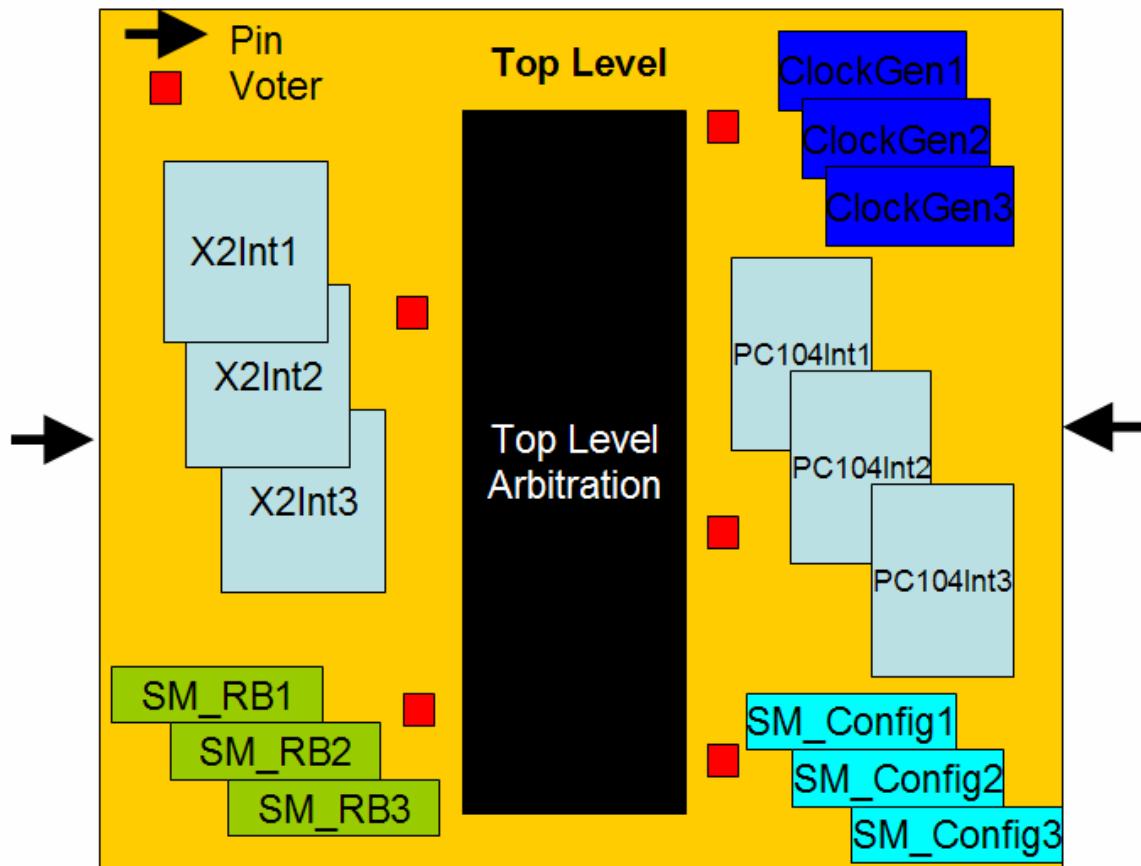


Figure 17. MFTX1 Implementation

The hypothesis with this design is that the Xilinx ISE software will not be able to distinguish between the redundant modules and the un-intentioned simplification will be less than in the BFTX1 implementation.

The same considerations mentioned for the implementation of BFTX1 apply in this case too.

The redundancy of components is a very effective technique of SEU mitigation that is easy to implement and can be performed entirely within a single device as long as the original design does not utilize more than a third of the total device [46].

C. CHAPTER SUMMARY

In this chapter, a review of the principal Fault Tolerant techniques applicable to FPGAs was presented. Three different implementations of a Fault Tolerant Control Unit were discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. EVALUATION OF THE FAULT TOLERANT CONTROL FPGA (FTX1)

In the previous chapter, three different implementation of a FTX1 were proposed. All three designs are based on TMR with some variations. These variations are a consequence of restrictions in the CFTP board or unknowns about the way in which Xilinx ISE software implements the designs.

The problem with the implementation of the 3PFTX1 design is its requirement of three pins per input and output signal that could not be accomplished with the current CFTP hardware. For this reason the 3PFTX1 was not implemented and its analysis is not covered in this work. The other two designs proposed in this thesis were implemented successfully in the CFTP. An analysis of BFTX1 and MFTX1 is presented in this chapter.

In order to evaluate the proposed FTX1 designs, a point of reference and specific benchmarks are needed. The best way to evaluate the Fault Tolerance of FTX1 against SEUs is to expose it to radiation and measure the effective cross section of the device but this option is expensive and was not available at the time of development of this work. A second option is to implement a component in FTX1 capable of injecting errors and execute scrubbing operations (as X1 does with X2). This job can be accomplished by including the functionality of X1 in the C&DH unit but this limits the portability of the CFTP board since in some applications there will not be a dedicated microprocessor to attend the FTX1 requirements and if there is one, the scrubbing process for X1 will reduce its performance. Another approach is the implementation of the functionality of X1 inside X2 but this sounds much like a self-licking ice cream cone. A third approach is the addition of a component (like an Anti-fuse FPGA) in the CFTP board that executes the injection of errors and scrubbing of X1 just for testing purposes but this implies modifications in the CFTP architecture and that was not an option.

Given the limitations for testing the FTX1, a manual injection of errors was implemented. This consists of the insertion of inverters at the outputs of one of the redundant modules in such a way that one out of three modules is always in error.

A. HARDWARE UTILIZATION IN THE FTX1 IMPLEMENTATIONS

The best approach to verify that the TMR was implemented properly in the FPGA by the Xilinx ISE software is to analyze the amount of logic being used by every design and that its functionality stays the same. The point of reference for this analysis is the original X1 and the benchmarks are the amount of logic included in each design and the evaluation of the functionality of FTX1.

The configuration data for the XQVR600 FPGA is 3,608,000 bits [25]. This information is stored in registers distributed in the different components of the FPGA. Table 4 shows the estimated distribution of configuration bits. One important consideration for the evaluation of the FTX1 is that the amount of configuration data in the FPGA is the same no matter the design of FTX1. SEUs occurring in the configuration bits do not always produce a detectable error. The concern for the reliability of the FTX1 is the configuration bits that if altered by a SEU will produce an error in the output data. The amount of logic being used by a FTX1 implementation determines the configuration bits susceptible to SEUs that will produce these hard errors.

RESOURCE	CONFIGURATION LATCHES	PERCENTAGE
CLB	103,086	0.35%
IOB	2165	0.06%
LUT	20409	5.61%
BRAM	135,300	3.75%
ROUTING	3,254,420	90.2%

Table 4. Distribution of Configuration bits in the XQVR 600 (From Ref. [33])

The majority of the configuration bits are used for the control of the routing resources so, errors in the interconnections are more likely to happen than in the logic components [33].

1. Contol FPGA (X1) Resources

The architecture of X1 was described in Chapter II. Table 5 shows the amount of logic actually used by every X1 component as is reported in the initial synthesis report.

Module	FF	Counter	Add/Sub	Comparator	Mux	FSM
Top_level	11	4	---	1	40	---
PC104Int	37	1	---	2	---	---
X2Int	166	3	2	3	8	---
Selectmap_RB	138	---	5	10	8	1
Selectmap_Config	113	1	3	9	---	---
ClockGen	2	1	---	2	---	---
TOTAL	467	10	10	27	56	1

Table 5. X1 Resources by Module

The synthesis report describes the amount of logic required by every component in order to implement its functionality as it is defined by the designer. But this logic can be optimized by the Xilinx ISE tools and the amount of logic used to physically implement a design can be much different than first reported. The final synthesis report given by Xilinx ISE contains the real amount of logic for the implementation of the design. Both numbers should be the same, but, since some optimization occurs in the synthesis process that can not be controlled, the amount of logic is simplified. This optimization does not affect the functionality of X1. Table 6 shows the amount of logic used in X1 reported in the final synthesis report.

Resource	Logic from Initial Synthesis report	Logic from Final Synthesis report	Difference between Reports
Finite State Machine (FSM)	1	1	0
D-type Flip Flop	467	159	308
Counter	10	6	4
Adder/Subtractor	10	13	-3
Comparator	27	27	0
Multiplexor	56	8	44

Table 6. Synthesis Reports for X1

2. Fault Tolerant Contol FPGA (BFTX1) Resources

The implementation of the BFTX1 is based on the TMR of instances of each component defined in top_level.vhd. The components being instantiated and triplicated by BTX1 are copies of the original X1 modules. In consequence the amount of resources used for each single component of BFTX1 is the same as X1 (Table 5). In addition to the X1 components and their triplicated instances, voters were included for the outputs of every triplicated module. The resources used by BFTX1 are shown in Table 7. The synthesis report contains the logic used by the components declared in BFTX1. Since the triplication in this case is on the instances, then just one component appears in the synthesis reports but in reality, when the FTX1 is configured in the FPGA, all triplicated instances are implemented and these resources appear in the final synthesis report.

Module	FF	Counter	Add/Sub	Comparator	Mux	FSM
Top_level	11	4	---	1	40	---
PC104Int	37	1	---	2	---	---
Voter_PC104Int	22	---	---	6	---	---
X2Int	166	3	2	3	8	---
Voter_X2Int	16	---	---	3	---	---
Selectmap_RB	138	---	5	10	8	1
Voter_SM_RB	47	---	---	9	---	---
Selectmap_Config	113	1	3	9	---	---
Voter_SM_Config	46	---	---	9	---	---
ClockGen	2	1	---	2	---	---
Cg_voter	6	---	---	---	---	---
TOTAL	1071	10	10	54	56	1

Table 7. BFTX1 Resources by Module

The reliability of BFTX1 is based on the successful triplication of the logic included in every component. If the optimization of the synthesis process reduces the TMR resources then the reliability of the design decreases. Table 8 illustrates the resources utilized for the implementation of BFTX1 as shown by the synthesis reports.

Resource	Logic from Initial Synthesis report	Logic from Final Synthesis report	Difference Between Reports
Finite State Machine (FSM)	1	1	0
D-type Flip Flop	1071	566	505
Counter	10	14	-4
Adder/Subtractor	10	35	-25
Comparator	54	106	-52
Multiplexor	56	12	44

Table 8. Synthesis Reports for BFTX1

Given that the instances inside top_level.vhd are triplicated, it is obvious to observe an increment in the resources being used but some comments need to be made here. Given that the original components of X1 are simplified, the same thing is happening with BFTX1 components and the triplication of the resources of X1 is taking place at the level of the already simplified logic.

3. Fault Tolerant Contol FPGA (MFTX1) Resources

The MFTX1 is based on the triplication of the components of X1 (except top level) instead of the triplication of instances. This means that the simplification already seen in the components of the original X1 and BFTX1 is occurring here too. In the initial synthesis report the logic of each of the triplicated components is shown. Voters are included at the output of the triplicated components. Table 9 shows the amount of logic utilized by this design.

Module	FF	Counter	Add/Sub	Comparator	Mux	FSM
Top_level	11	4	---	1	40	---
PC104IntArm1	37	1	---	2	---	---
PC104IntArm2	37	1	---	2	---	---
PC104IntArm3	37	1	---	2	---	---
Voter_PC104Int	22	---	---	6	---	---
X2Int1	166	3	2	3	8	---
X2Int2	166	3	2	3	8	---

Module	FF	Counter	Add/Sub	Comparator	Mux	FSM
X2Int3	166	3	2	3	8	---
Voter_X2Int	16	---	---	3	---	---
Selectmap_RB1	138	---	5	10	8	1
Selectmap_RB2	138	---	5	10	8	1
Selectmap_RB3	138	---	5	10	8	1
Voter_SM_RB	47	---	---	9	---	---
Selectmap_Config1	113	1	3	9	---	---
Selectmap_Config2	113	1	3	9	---	---
Selectmap_Config3	113	1	3	9	---	---
Voter_SM_Config	46	---	---	9	---	---
ClockGen1	2	1	---	2	---	---
ClockGen2	2	1	---	2	---	---
ClockGen3	2	1	---	2	---	---
Cg_voter	6	---	---	---	---	---
TOTAL	1516	22	30	106	88	3

Table 9. MFTX1 Resources by Module

Table 10 illustrates the difference between the resources reported in the initial and final synthesis reports. As can be seen, the initial reports from BFTX1 and MFTX1 are different but the amount of logic shown in the final reports is the same for both designs. This means that indeed the simplification from the initial to the final synthesis resources is a smart move and instead of deleting all replicated logic, the software is replacing some components by others or it is sharing resources between the different components, deleting the unnecessary resources.

B. EVALUATION OF THE FTX1 IMPLEMENTATIONS THROUGH HARDWARE UTILIZATION

The initial point of analysis is the original X1 design. This unit contained a certain amount of logic. Adders, subtractors, comparators and other components are generated whenever a process in X1 requires them and the different signals used by the unit are saved in registers. Then the synthesis tool determines which components can be deleted,

replaced or shared. One example is a signal that does not change values inside a module. This signal is initially saved in a register but once the synthesis tool recognizes that there is no change in the value of the signal, it replaces it with a value in a register to save resources. Another example is that instead of generating a 4-to-1 multiplexer for each process requiring one, the synthesis tool combines the functionality of small multiplexers and generates a larger one that simplifies the logic. So in reality, the simplification of resources tries to increase the performance of the design and to save resources for future utilization. Even though there is an unintentional simplification of resources, X1 has proved to be a functional and efficient component in the past but no reliability test had been applied on it.

Resource	Logic from Initial Synthesis report	Logic from Final Synthesis report	Difference Between Reports
Finite State Machine (FSM)	1	1	0
D-type Flip Flop	1516	566	1050
Counter	22	14	8
Adder/Subtractor	30	35	-5
Comparator	106	106	0
Multiplexor	88	12	76

Table 10. Synthesis Reports for MFTX1

The inconvenience is that components that need to be triplicated in order to implement a complete TMR are being simplified and there is no control about this process.

Three different Fault Tolerant Control Units were proposed in Chapter V. 3PFTX1 could not be implemented with the current CFTP hardware and its analysis is not addressed but it is worth mentioning that this is the ideal model to implement a complete TMR of the original X1.

BFTX1 is the second implementation proposed. Its functionality and implementation was discussed in previous sections. The important point to mention in this analysis is that BFTX1 was produced from the TMR of instances of the original X1

components, and since these components are simplified by the Xilinx software before any triplication takes place, the same behavior is observed in BFTX1 components. Since simplification is happening at the original X1 components without control, the amount of simplification occurring in the BFTX1 is an unknown too. From Table 10 is evident that the original X1 is being triplicated. In fact, the instances being triplicated are based on the already simplified components.

BFTX1 and MFTX1 have a lot in common. Their basic difference is that the first design bases its reliability in the TMR of the instantiation of components and the second design in the TMR of the components before being instantiated. The purpose of MFTX1 was to determine if some un-intentioned simplification was occurring in the BFTX1 by comparing the resources utilized by both designs. The hypothesis was that since instantiations of the same component are being used in BFTX1, Xilinx ISE synthesis tool was smart enough to recognize the redundancy of the instances from the same component and in consequence simplified the design reducing the redundant resources and the reliability being added to FTX1.

MFTX1 design triplicates the components instead of the instantiations. Table 11 shows the resources used by the different FTX1 implementations. The amount of logic used in MFTX1 is basically three times more than the original X1 and pretty much the same as the BFTX1 approach.

Resource	X1	BFTX1	MFTX1
Equivalent Gates	32,542	96,891	95,878
Slices	815 (11%)	2418 (34%)	2332 (33%)
Slice FFs	735 (5%)	2100 (15%)	2093 (15%)
4-input LUTs	1354 (8%)	4060 (29%)	3952 (28%)
IOBs	121 (72%)	121 (72%)	121 (72%)

Table 11. Resources of FTX1 Implementations

The Xilinx FPGAs and the software tools used to program them have secrets. This hidden information limits the implementation of a complete TMR and the control over the optimization process of the logic being used by a design.

The bottom line is that in the original X1 some simplification is occurring and the TMR implementations BFTX1 and MFTX1 are using these simplified components as replicated instances or redundant components. This proves that the TMR is being implemented properly and since the optimization of the original components of X1 does not degrade its functionality, the same fact can be expected in the FTX1 designs.

C. EVALUATION OF FUNCTIONALITY AND PERFORMANCE OF THE FTX1

Two of the basic requirements for the implementation of a Fault Tolerant system are that the functionality and performance of the original components stay the same or present only slight changes. In the previous section it was demonstrated that indeed the TMR is being properly implemented in both BFTX1 and MFTX1 designs. But this analysis is not enough to prove that the functionality and performance of the original X1 do not suffer modifications. From Chapter III, the basic functions of X1 are:

- Control of the initial configuration and subsequent reconfigurations of X2.
- Interfacing with system memory (Flash and EEPROM).
- Handling communication with the outside world through a PC/104 bus.
- Verification of the experiment running in X2 through scrubbing.

In order to show that the FTX1 designs do not alter the original functionality of X1 each of the functions mentioned above needs to be verified. In Reference [55] the process of design and implementation of experiments on the CFTP board are explained. The best way to verify that the FTX1 designs work properly is to program the CFTP board using FTX1 instead of X1 and verify that the outputs of X2 are the same.

The first step is programming X1 to write an experiment's configuration to the flash memory. After this step is completed, a message is received from the CFTP informing that X1 was programmed successfully. The only function of this configuration is to load the flash memory and after this task is completed a new X1 configuration is loaded as explained in the following paragraphs. The period of time that this initial configuration of X1 is in use is short therefore fault tolerance is not implemented on it.

The second step in the process of programming the CFTP board consists of writing the configuration of an experiment to the flash.

The third step consists of programming X2 with the configuration stored in the flash.

The last step consists of programming X1 again. This configuration is the basis of the development of this work. If X1 is working properly, it must be able to read the configuration and the output of X2, pass that information to the PC/104 bus and read the original configuration from the flash. This X1 configuration works through all the operation of the CFTP until a new experiment needs to be loaded in X2.

The evaluation of the operation of X1 is merely based on the analysis of the output of the experiment running on X2. If the output of X2 is the same for different configurations of X1, we can assume that indeed the functionality of X1 did not present any variation.

Each design of X1 is related to one specific experiment running on X2 and every time a new experiment is loaded in X2, some modifications need to be done to the X1 code. In this case, the experiment used with the FTX1 designs implemented in this work is a Multiplier developed by Gerald W. Caldwell [55].

After the CFTP is in full operation X1 takes the outputs from the experiment running on X2 and sends that information to the PC/104 bus. This is the first point in the evaluation of X1. The next point is the evaluation of the correctness of the output from X2.

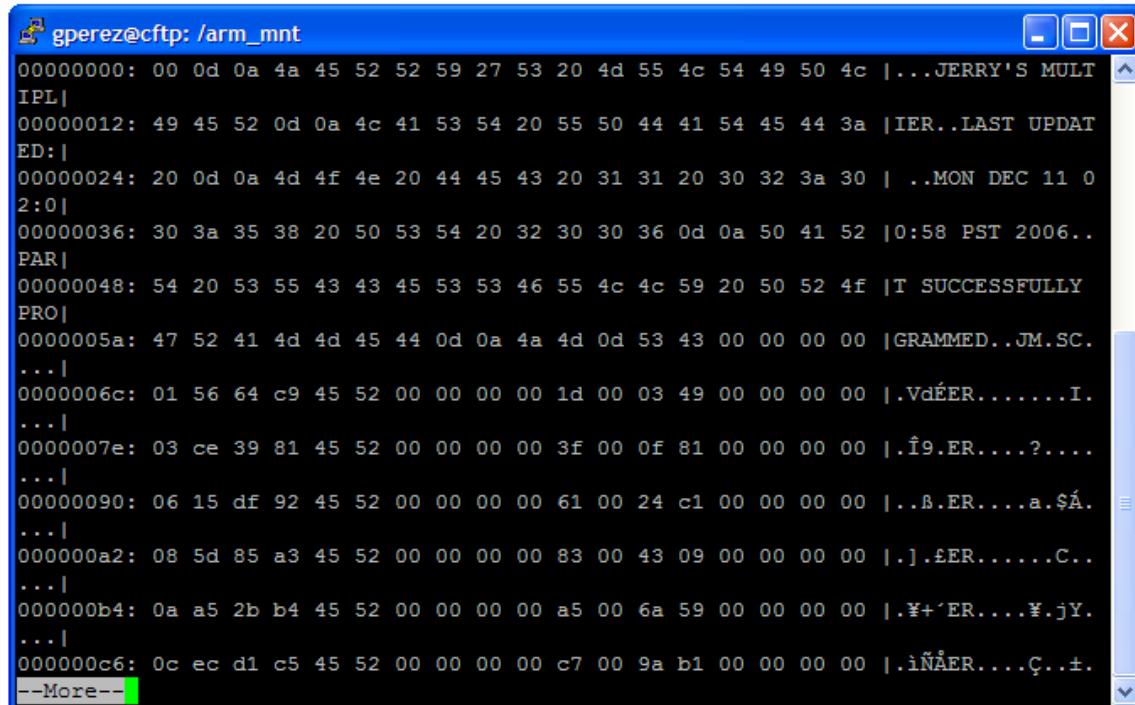
A complete explanation of the design and operation of this multiplier is presented in Reference [55]. What has to be observed at the output of X2 is a sequence of 18 hexadecimal numbers arranged in the following sequence:

- The letter “E” (hex 45).
- The letter “R” (hex 52).
- A sequence of eight zeros.
- The 2-digit output of a counter that provides the inputs to the multiplier.

- The 4-digit output of the multiplier that should be the square of the counter input.
- A time stamp.

The letters E, R and the sequence of zeros are constant in every output but the time stamp and the count of the multiplier vary.

The output from X2 using the original X1 is shown in Figure 18.



```

gperez@cftp: /arm_mnt
00000000: 00 0d 0a 4a 45 52 52 59 27 53 20 4d 55 4c 54 49 50 4c |...JERRY'S MULT
IPL|
00000012: 49 45 52 0d 0a 4c 41 53 54 20 55 50 44 41 54 45 44 3a |IER..LAST UPDAT
ED:|
00000024: 20 0d 0a 4d 4f 4e 20 44 45 43 20 31 31 20 30 32 3a 30 | ..MON DEC 11 0
2:0|
00000036: 30 3a 35 38 20 50 53 54 20 32 30 30 36 0d 0a 50 41 52 |0:58 PST 2006..
PAR|
00000048: 54 20 53 55 43 43 45 53 53 46 55 4c 4c 59 20 50 52 4f |T SUCCESSFULLY
PRO|
0000005a: 47 52 41 4d 4d 45 44 0d 0a 4a 4d 0d 53 43 00 00 00 00 |GRAMMED..JM.SC.
...|
0000006c: 01 56 64 c9 45 52 00 00 00 00 1d 00 03 49 00 00 00 00 |.VdÉER.....I.
...|
0000007e: 03 ce 39 81 45 52 00 00 00 00 3f 00 0f 81 00 00 00 00 |.í9.ER....?.....
...|
00000090: 06 15 df 92 45 52 00 00 00 00 61 00 24 c1 00 00 00 00 |..B.ER....a.ŚA.
...|
000000a2: 08 5d 85 a3 45 52 00 00 00 00 83 00 43 09 00 00 00 00 |.].ŁER.....C..
...|
000000b4: 0a a5 2b b4 45 52 00 00 00 00 a5 00 6a 59 00 00 00 00 |.Ỳ+`ER....Ỳ.jY.
...|
000000c6: 0c ec d1 c5 45 52 00 00 00 00 c7 00 9a b1 00 00 00 00 |.iÑAER....ç...±.
--More-- 

```

Figure 18. Output from X2 through X1

```

gperez@cftp: /arm_mnt
00000000: 00 0d 0a 4a 45 52 52 59 27 53 20 4d 55 4c 54 49 50 4c |...JERRY'S MULTI
PL|
00000012: 49 45 52 0d 0a 4c 41 53 54 20 55 50 44 41 54 45 44 3a |IER..LAST UPDATE
D:|
00000024: 20 0d 0a 4d 4f 4e 20 44 45 43 20 31 31 20 30 32 3a 31 | ..MON DEC 11 02
:1|
00000036: 38 3a 31 34 20 50 53 54 20 32 30 30 36 0d 0a 50 41 52 |8:14 PST 2006..P
AR|
00000048: 54 20 53 55 43 43 45 53 53 46 55 4c 4c 59 20 50 52 4f |T SUCCESSFULLY P
RO|
0000005a: 47 52 41 4d 4d 45 44 0d 0a 4a 4d 0d 53 43 00 00 00 3b |GRAMMED..JM.SC..
.:|
0000006c: a9 45 52 00 00 00 00 00 1b 00 02 d9 00 00 00 00 02 b2 d4 |@ER.....Ù.....
*Ó|
0000007e: a0 45 52 00 00 00 00 00 3d 00 0e 89 00 00 00 00 04 fa 7a | ER....=.....
úz|
00000090: b1 45 52 00 00 00 00 00 5f 00 23 41 00 00 00 00 07 42 20 |±ER...._.#A.....
B |
000000a2: c2 45 52 00 00 00 00 81 00 41 01 00 00 00 00 09 89 c6 |ÀER.....A.....
.E|
000000b4: d3 45 52 00 00 00 00 a3 00 67 c9 00 00 00 00 0b d1 6c |ÓER....£.gÉ.....
Ñ1|
000000c6: e4 45 52 00 00 00 00 c5 00 97 99 00 00 00 00 0e 19 12 |äER....À.....
--More-- 

```

Figure 19. Output from X2 through BFTX1

```

gperez@cftp: /arm_mnt
00000000: 00 0d 0a 4a 45 52 52 59 27 53 20 4d 55 4c 54 49 50 4c |...JERRY'S MULTI
PL|
00000012: 49 45 52 0d 0a 4c 41 53 54 20 55 50 44 41 54 45 44 3a |IER..LAST UPDATE
D:|
00000024: 20 0d 0a 4d 4f 4e 20 44 45 43 20 31 31 20 30 32 3a 33 | ..MON DEC 11 02
:3|
00000036: 32 3a 34 32 20 50 53 54 20 32 30 30 36 0d 0a 50 41 52 |2:42 PST 2006..P
AR|
00000048: 54 20 53 55 43 43 45 53 53 46 55 4c 4c 59 20 50 52 4f |T SUCCESSFULLY P
RO|
0000005a: 47 52 41 4d 4d 45 44 0d 0a 4a 4d 0d 53 43 00 00 00 00 |GRAMMED..JM.SC..
.:|
0000006c: 00 d0 81 18 45 52 00 00 00 00 1c 00 03 10 00 00 00 00 |.Ð..ER.....|.
..|
0000007e: 03 48 55 91 45 52 00 00 00 00 3e 00 0f 04 00 00 00 00 |.HU.ER....>.....
..|
00000090: 05 8f fb a2 45 52 00 00 00 00 60 00 24 00 00 00 00 00 |..ûçER....`.$...
..|
000000a2: 07 d7 a1 b3 45 52 00 00 00 00 82 00 42 04 00 00 00 00 |..×;³ER.....B...
..|
000000b4: 0a 1f 47 c4 45 52 00 00 00 00 a4 00 69 10 00 00 00 00 |..GÄER....¤.i...
..|
000000c6: 0c 66 ed d5 45 52 00 00 00 00 c6 00 99 24 00 00 00 00 |.fiÓER....£..$...
--More-- 

```

Figure 20. Output from X2 through MFTX1

There is not a real difference from the point of view of functionality and resource utilization between BFTX1 and MFTX1. Although the automatic optimization and the TMR applied to the original X1, there is no modification in the functionality of the Control FPGA.

Another way to verify the performance of the designs is through the timing report. The maximum frequency at which BFTX1 and MFTX1 can run is about 59.563 MHz and since the clock used by the CFTP no matter the X1 design used is 51 MHz there is no detriment in performance.

D. MANUAL INJECTION OF ERRORS

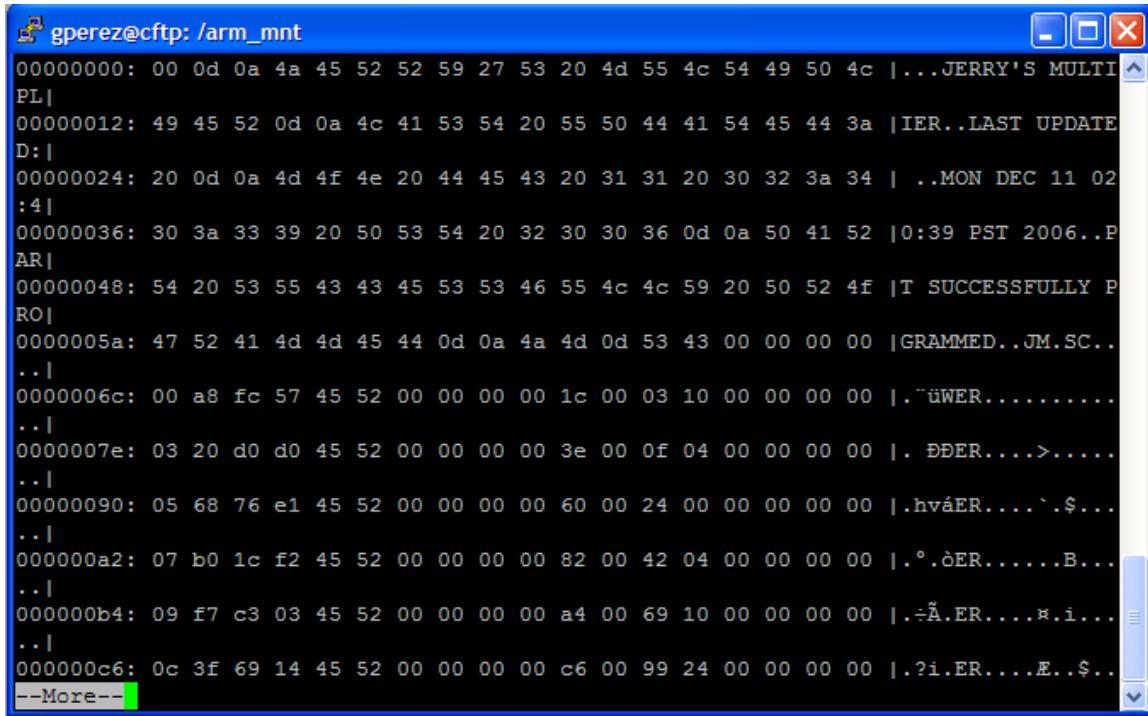
After proving that the BFTX1 and MFTX1 designs were implemented correctly in the Control FPGA and there is no change in functionality on the original X1, the next step was to test the designs for fault tolerance.

The only method currently available for testing the FTX1 is the manual injection of errors. As explained in Chapter V, this method consists of the modification of one of the triplicated modules in such a way that its outputs differ from the outputs of the other two modules.

The best way to implement the manual injection is to place inverters at the output of the faulty module (Figure 18). Consequently, these outputs are always different than the outputs of the other two modules.

In the next step, the outputs coming from the triplicated modules are voted by a majority voter and the final output of the TMR is the correct value.

This method is easy to realize and provides a better means for the evaluation of the FTX1 designs than just the evaluation of the correct implementation of TMR. Figure 21 shows the output of X2 using MFTX1.



gperez@cftp: /arm_mnt

```
00000000: 00 0d 0a 4a 45 52 52 59 27 53 20 4d 55 4c 54 49 50 4c |...JERRY'S MULTI
PL|
00000012: 49 45 52 0d 0a 4c 41 53 54 20 55 50 44 41 54 45 44 3a |IER..LAST UPDATE
D:|
00000024: 20 0d 0a 4d 4f 4e 20 44 45 43 20 31 31 20 30 32 3a 34 | ..MON DEC 11 02
:4|
00000036: 30 3a 33 39 20 50 53 54 20 32 30 30 36 0d 0a 50 41 52 |0:39 PST 2006..P
AR|
00000048: 54 20 53 55 43 43 45 53 53 46 55 4c 4c 59 20 50 52 4f |T SUCCESSFULLY P
RO|
0000005a: 47 52 41 4d 4d 45 44 0d 0a 4a 4d 0d 53 43 00 00 00 00 |GRAMMED..JM.SC..
...|
0000006c: 00 a8 fc 57 45 52 00 00 00 00 1c 00 03 10 00 00 00 00 |.^iWER.....
...|
0000007e: 03 20 d0 d0 45 52 00 00 00 00 3e 00 0f 04 00 00 00 00 |. ððER....>.....
...|
00000090: 05 68 76 e1 45 52 00 00 00 00 60 00 24 00 00 00 00 00 |.hváER....`.$...
...|
000000a2: 07 b0 1c f2 45 52 00 00 00 00 82 00 42 04 00 00 00 00 |.º.òER....B...
...|
000000b4: 09 f7 c3 03 45 52 00 00 00 00 a4 00 69 10 00 00 00 00 |.¬Ã.ER....¤.i...
...|
000000c6: 0c 3f 69 14 45 52 00 00 00 00 c6 00 99 24 00 00 00 00 |.?.i.ER....£..$...
--More--
```

Figure 21. Output from X2 through MFTX1 with Manual Injection of Errors

Effects of SEUs over FPGAs were analyzed in previous chapters. One of the limitations of the manual injection of errors is that not all the SEUs' effects can be simulated. In fact, only errors in logic components can be simulated since there is no current way to modify the configuration bits to test the FTX1 designs against configuration errors with the use of manual injection.

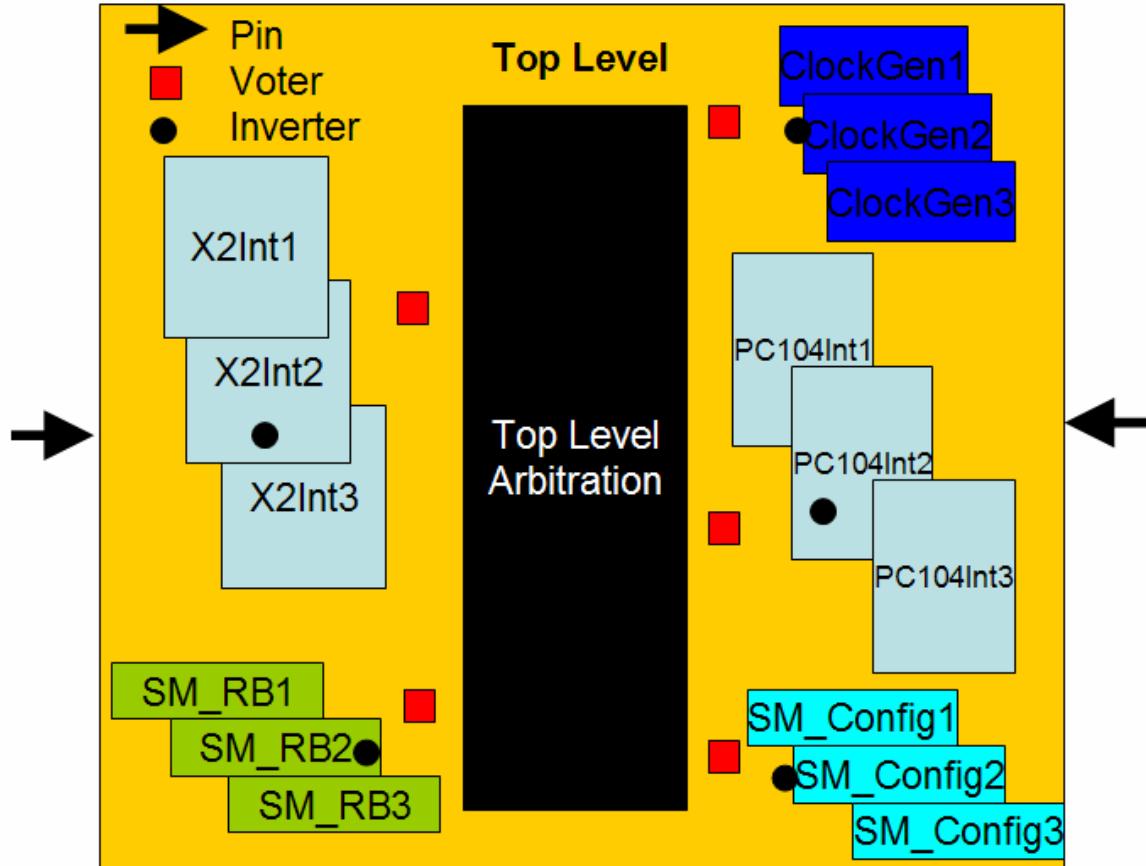


Figure 22. Manual Injection of Errors in MFTX1

Resource	X1	BFTX1	BFTX1_Inj	MFTX1	MFTX1_Inj
Equivalent Gates	32,502	96,891	95,987	95,878	96,673
Slices	815 (11%)	2418 (34%)	2352 (34%)	2332 (33%)	2346 (33%)
Slice FFs	735 (5%)	2100 (15%)	2098 (15%)	2093 (15%)	2087 (15%)
4-input LUTs	1354 (8%)	4060 (29%)	3934 (29%)	3952 (28%)	4044 (29%)
IOBs	121 (72%)	121 (72%)	121 (72%)	121 (72%)	121 (72%)

Table 12. Resources of FTX1 Implementations with Manual Injection

The resource utilization of BFTX1 and MFTX1 designs changed with the implementation of manual injection, but the changes were minor. Table 12 shows the current resources utilized by each design.

E. CHAPTER SUMMARY

In this chapter, different implementations were evaluated from the hardware utilization point of view to prove that indeed the Xilinx ISE software implemented the TMR of the designs properly. An analysis of the outputs from X2 utilizing the different X1 designs and the testing of the FTX1 designs through manual injection of errors were also evaluated.

VII. CONCLUSIONS AND RECOMMENDATIONS

This thesis described the effects of the space environment on electronics and specifically on FPGAs. The Fault Tolerant techniques applicable to FPGAs were then explained. Three different implementations of a Fault Tolerant Control Unit (FTX1) within an FPGA for space applications were proposed. Finally, an analysis of the way in which TMR is being implemented by the Xilinx tools, the analysis of functionality of the FTX1 designs and manual injection of errors to test the designs were presented.

A. SUMMARY

The harsh space environment represents a challenge for the implementation of computer systems for space applications. SEUs are the biggest concern when a SRAM-based FPGA is a component of those systems. Circuit-Level and High-Level modification methods have been proposed for the mitigation of SEUs in FPGAs. TMR represents the most reliable technique to implement a Fault Tolerant Control Unit. Three different designs of a FTX1 were proposed: 3PFTX1, BFTX1 and MFTX1. All these designs are based on the TMR approach with slight variations.

3PFTX1 represents the complete TMR of the original X1. This implementation consists in the triplication of the resources of X1 from inputs to outputs. By definition it is the only way to protect all the components of X1 including the configuration bits. The requirement of 3 pins per input and output signal was a restriction for the successful implementation of this design in the XQVR600-4CB228 FPGA.

BFTX1 was then proposed to solve that limitation. This implementation is based on the triplication of instances of the different components of X1 except top level. In this design the TMR is not complete and there are single points of failure present given that the input and output signals and the UCF file can not be replicated. Some unintended optimization was observed.

MFTX1 was proposed to determine the source of the automatic optimization that the Xilinx tools apply to the TMR implementation. Components instead of instances were triplicated. The amount of resources utilized by BFTX1 was pretty much the same as the resources utilized in MFTX1 disproving the hypothesis that the triplication of instances was the cause of the optimization.

An analysis of the files generated after the implementation of X1, BFTX1 and MFTX1 in the XQVR600-4CB228 showed that the unintentional optimization of resources was not a consequence of the redundancy of components but a consequence of the architecture of the original X1 components. Table 13 shows the resource utilization for each of the X1 implementations.

Resource	X1	BFTX1_Inj	MFTX1_Inj
Equivalent Gates	32,502	95,987	96,673
Slices	815 (11%)	2352 (34%)	2346 (33%)
Slice FFs	735 (5%)	2098 (15%)	2087 (15%)
4-input LUTs	1354 (8%)	3934 (29%)	4044 (29%)
IOBs	121 (72%)	121 (72%)	121 (72%)

Table 13. Resources of FTX1 Implementations

After proving that the FTX1 designs were implemented correctly in the CFTP board the next step was the evaluation of the functionality of these designs. It was done through the analysis of the outputs obtained from X2 utilizing each of X1 designs.

Finally a testing through manual injection of errors was implemented. Even this is not an optimal evaluation technique it served as a starting point to prove that the TMR is working properly and that in fact the reliability of X1 is increased.

B. CONCLUSIONS

Different Fault Tolerant Techniques have been proposed for the implementation of designs based on SRAM FPGAs for space applications but the TMR is indeed the only method that completely protects these designs.

3PFTX1 represents the ideal design of a Fault Tolerant Control Unit within an FPGA that is the objective of this work but the hardware of the CFTP board restricts its implementation.

Given that a FTX1 with complete TMR is not an option for the current CFTP, BFTX1 and MFTX1 designs were proposed. These designs implement successfully a Fault Tolerant Control Unit through the triplication of most of the components of X1. Single points of failure are present in these designs and this can only be corrected through modifications in the hardware of the CFTP board.

The automatic and un-intentional optimization implemented by Xilinx tools does not harm the functionality of X1 and the reliability provided by the triplication of the simplified components of X1.

The FTX1 designs proved not to harm the functionality and performance of the original X1 that are basic requirements for the implementation of a Fault Tolerant System.

It was proved through manual injection of errors that indeed the TMR improves the reliability of the Control FPGA.

C. FOLLOW-ON RESEARCH

The complete implementation of TMR in the Fault Tolerant Control Unit is the only way to ensure that every component inside X1 including the configuration memory will be protected against SEUs. Modifications in the CFTP board and specifically, the replacement of the current XQVR600-4CB228 FPGA for one with three times more pins need to be done in order to implement a completely reliable X1.

Both BFTX1 and MFTX1 designs triplicate all X1 components but the top_level.vhd module. This module could not be triplicated because of the lack of pins on the chip. A way to at least triplicate the functionality inside this module needs to be investigated.

The evaluation of the designs proposed for the implementation of a FTX1 is based on the resources utilized by each design and the comparison of the outputs of X2 controlled by them. The best approach to evaluate the improvement of the reliability of

X1 is through radiation testing of the device. It has to be accomplished with the purpose of calculating the effective cross section, the increment of sensitive bits to SEUs that produce a persistent error and the behavior and ability to detect and correct errors in each design.

An additional hardware component (Anti-fuse FPGA) in the CFTP board or a C&DH with software tools need to be implemented in order to be able to scrub and inject errors inside the FTX1 as X1 does with X2.

LIST OF REFERENCES

- [1] Hiroyuki Yashiro, Yoshiro Takahashi, Teruo Fujiwara, "Verification of Assurance of Space On-board Distributed Computer System," in *Proc. IEEE 6th Intl. Symp. on High Assurance Engineering*, 2001.
- [2] "Space Environments & Effects Program," NASA Marshall Space Flight Center, (website) <http://www.eas.asu.edu/~holbert/eee460/spacerad.html>, October 2006.
- [3] "Solar Connections: A Science Initiative for NASA Space Physics," (website) http://umbra.nascom.nasa.gov/solar_connections/domain.html, October 2006.
- [4] "European Space Agency, Space Environment and Effects Analysis Section," (website) <http://www.eas.asu.edu/~holbert/eee460/spacerad.html>, October 2006.
- [5] "The NASA ASIC Guide: Assuring ASICS for Space," (website) <http://parts.jpl.nasa.gov/asic/Sect.3.4.html#A0>, October 2006.
- [6] "Wikipedia," (website) <http://en.wikipedia.org>, November 2006.
- [7] "Single-event effects in FPGAs," (website) <http://www.actel.com/documents/FirmErrorPIB.pdf>, October 2006.
- [8] I. Mouret, M. Allenspach, R.D. Schrimpf, J.R. Brews, K.F. Galloway, P. Calvel, "Temperature and angular dependence of substrate response in SEGR," in *IEEE Trans. on Nuclear Science, Vol. 41, Issue 6, Part 1, pp. 2216-2221*, December 1994.
- [9] Dean A. Ebert, "Design and Development of a Configurable Fault Tolerant Processor (CFTP)," Master's Thesis, Naval Postgraduate School, Monterey, California, June 2003.
- [10] James C. Coudeyras, "Radiation Testing of the Configurable Fault Tolerant Processor (CFTP) For Space-Based Applications," Master's Thesis, Naval Postgraduate School, Monterey, California, December 2005.
- [11] Steven A. Johnson. "Implementation of a Configurable Fault Tolerant Processor (CFTP)," Master's Thesis, Naval Postgraduate School, Monterey, California, March 2003.
- [12] Peter J. Majewicz, "Implementation of a Configurable Fault Tolerant Processor (CFTP) Using Internal Triple Modular Redundancy (TMR)," Master's Thesis, Naval Postgraduate School, Monterey, California, December 2005.
- [13] Malgorzata Chrzanowska-Jeske, "Architecture and Technology of FPGAs – An Overview," Northcon/93 Conference Record, pp. 82-86, October 1993.

- [14] Stephen Brown, Jonathan Rose, “Architecture of FPGAs and CPLDs: A Tutorial,” in *IEEE Design and Test of Computers, Vol. 13, No 2 pp. 42-57*, 1996.
- [15] J. J. Wang, Brian Cronquist, John McCollum, Huan Tseng, Roy Lambertson, Stefan Goethe, Rich Katz, Igor Kleyner, “Radiation Effects on Flash Memory Based FPGA,” in MAPLD International Conference 2000, September 2000.
- [16] Xilinx Corp., “VirtexTM 2.5 V Field Programmable Gate Arrays,” App. Note DS003-2, December 2002.
- [17] Joe Fabula, Howard Bogrow. “Total Ionizing Dose Performance of SRAM-based FPGAs and supporting PROMs,” in MAPLD International Conference 2000, September 2000.
- [18] D.M. MacQueen, D.M. Gingrich, N. J. Buchanan and P.W. Green, “Total Ionizing Dose Effects in a SRAM-Based FPGA,” in *IEEE Radiation Effects and Data Workshop, pp. 24-29*, July 1999.
- [19] Michael Caffrey, Paul Graham, Eric Johnson, Michael Wirthlin, Carl Carmichael, “Single-Event Upsets in SRAM FPGAs,” MAPLD International Conference 2002, September 2002.
- [20] M. Alderighi, A. Candelori, F. Casini, S. D’Angelo, M. Mancini, A. Paccagnella, S. Pastore, G.R. Sechi, “Heavy Ion Effects on Configuration Logic of Virtex FPGAs,” in *IEEE 11th Proc. Intl. On-line Testing Symposium, pp.49-53*, July 2005.
- [21] M. Bellato, P. Bernardi, D. Bortolato, A. Candelori, M. Ceschia, A. Paccagnella, M. Rebaudengo, M. Sonza Reorda, M. Violante and P. Zambolin, “Evaluating the effects of SEUs affecting the configuration memory of a SRAM-based FPGA,” in *IEEE Proc. of Design Automation and Test in Europe Conference and Exhibition, pp. 188-193*, December 2004.
- [22] M. Stettler, M. Caffrey, P. Graham, J. Krone, “Radiation effects and mitigation strategies for modern FPGAs,” in 10th Workshop on Electronicsfor LHC Experiments and Future Experiments, September 2004.
- [23] Jih-Jong Wang, Brian Cronquist, John McCollum, Wanida Parker, Rich Katz, and Igor Kleyner, “Radiation tolerant antifuse FPGA,” MAPLD International Conference 2002, September 2002.
- [24] Mindy Surratt, “CFTP Development Environment Technical Manual,” Technical Manual, Naval Postgraduate School, Monterey, California, December 2005.
- [25] Xilinx Corp., “Aerospace and Defense Programmable Logic Data Book,” July 2000.

- [26] Joshua D. Snodgrass, “Low-Power Fault Tolerance for Spacecraft FPGA-Based Numerical Computing,” PhD Dissertation, Naval Postgraduate School, Monterey, California, September 2006.
- [27] Xilinx Corp., “QPro XQR17V16 Radiation Hardened 16Mbit QML Configuration PROM,” App. Note DS126 (v1.0), December 2003.
- [28] “Intel® Advanced+ Boot Block Flash Memory,” (website)
<ftp://download.intel.com/design/flcomp/datashts/29064523.pdf>, November 2006.
- [29] “SEAKR Engineering SEE Radiation Data Report,” Technical Report, Centennial, Colorado, March 2002.
- [30] “PC/104 Specifications Version 2.5,” PC104 Embedded Consortium, November 2003.
- [31] Xilinx Corp., “Virtex FPGA Series Configuration and Readback,” App. Note XAPP138, March 2006.
- [32] Christopher P. Fuhrman, Sailesh Chutani, Henri J. Nussbaumer, “Hardware/Software Fault Tolerance with Multiple Task Modular Redundancy,” in *Proc. IEEE Symposium on Computers and Communications*, pp. 171-177, June 1995.
- [33] Earl Fuller, Michael Caffrey, Anthony Salazar, Carl Carmichael and Joe Fabula “Radiation Testing Update, SEU Mitigation, and Availability Analysis of the Virtex FPGA for Space Reconfigurable Computing,” MAPLD International Conference 2000, September 2000.
- [34] Kenneth A. LaBel, Charles E. Barnes, Paul W. Marshall, Cheryl J. Marshall, Allan H. Johnston, Robert A. Reed, Janet L. Barth, Christina M. Seidleck, Sammy A. Kayali, Martha V. O’Bryan, “A Roadmap for NASA’s Radiation Effects Research in Emerging Microelectronics and Photonics,” in *IEEE Aerospace Conference Proc.*, Vol. 5, pp. 535-545, March 2000.
- [35] Wally Gibbons and Harry Ames, “Use of FPGAs in Critical Space Flight Applications a Hard Lesson,” in MAPLD International Conference 1999, September 1999.
- [36] Brent Robertson and Eric Stoneking, “Satellite GN&C Anomaly Trends,” in AAS Guidance and Control Conference, 2003.
- [37] Abderrahim Doumar, Satoshi Kaneko and Hideo Ito “Defect and fault Tolerance FPGAs by Shifting the Configuration Data,” in *Proc. Intl. Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 377-385, November 1999.

- [38] Fernanda Lima Kastensmidt, Gustavo Neuberger, Luigi Carro and Ricardo Reis, “Designing and Testing Fault-Tolerant Techniques for SRAM-based FPGAs,” in Proc. 1st Conference on Computer Frontiers, pp. 419-432, April 2004.
- [39] John Lach, William H. Mangione-Smith and Miodrag Potkonjak, “Low Overhead Fault-Tolerant FPGA Systems,” in IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 6, Issue 2, pp. 212-221, June 1998.
- [40] Sergio D'Angelo, Cecilia Metra and Giacomo Sechi, “Transient and Permanent Fault Diagnosis for FPGA-Based TMR Systems,” in Intl. Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 330-338, November 1999.
- [41] Sandi Habinc, “Functional Triple Modular Redundancy (FTMR). VHDL Design Methodology for Redundancy in Combinatorial and Sequential Logic,” Gaisler Research, Design and Assessment Report (Version 0.2), December 2002.
- [42] Carl Carmichael, “Triple Module Redundancy Design Techniques for Virtex FPGAs,” Xilinx App. Note XAPP197, July 2006.
- [43] Praveen Kumar Samudrala and Jeremy Ramos, “Selective Triple Modular Redundancy (STMR) Based Single-Event Upset (SEU) Tolerant Synthesis for FPGAs,” in IEEE Trans. on Nuclear Science, Vol. 51, Issue 5, Part 4, pp. 2957-2969, October 2004.
- [44] Melanie Berg, “Fault tolerant implementation within SRAM based FPGA designs based upon the increased level of single even upset susceptibility,” in IEEE Proc. 12th IEEE Intl. Symposium on On-Line Testing, pp. 89-91, 2006.
- [45] Fernanda Lima, Luigi Carro and Ricardo Reis, “Reducing Pin and Area Overhead in Fault-Tolerant FPGA-based Designs,” Proc. 2003 ACM/SIGDA Intl. Symposium on FPGAs, pp. 108-117, 2003.
- [46] Ghazanfar Asadi and Mehdi B. Tahoori, “Soft Error Rate Estimation and Mitigation for SRAM-Based FPGAs,” Proc. 2005 ACM/SIGDA Intl. Symposium on FPGAs, pp. 149-160, February 2005.
- [47] Abderrahim Doumar and Hideo Ito, “Detecting, Diagnosing, and Tolerating Faults in SRAM-Based Field Programmable Gate Arrays: A Survey,” In IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 11, Issue 3, pp. 386-405, June 2003.
- [48] Paul Graham, Michael Caffrey, Jason Zimmerman, Prasanna Sundararajan, Eric Johnson, and Cameron Patterson, “Consequences and Categories of SRAM FPGA Configuration SEUs,” in MAPLD International Conference 2003, September 2003.
- [49] Xilinx Corp., “TMRTool User Guide (V6.2.03i),” September 2004.

- [50] S. D'Angelo, C. Metra, S. Pastore, A. Pogutz, G.R. Sechi, "Fault-Tolerant Voting Mechanism and Recovery Scheme for TMR FPGA-based Systems," In IEEE Proc. 1998 Intl. Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 233-240, November 1998.
- [51] J. Von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," in Automata Studies, Ann. of Math. Studies, No 34, pp. 43-98, 1956.
- [52] M. Alderighi, Member, A. Candelori, F. Casini, S. D'Angelo, M. Mancini, A. Paccagnella, S. Pastore, and G. R. Sechi, "SEU Sensitivity of Virtex Configuration Logic," in IEEE Trans. on Nuclear Science, Vol. 52, issue 6, Part 1, pp. 2462-2467, December 2005.
- [53] P. S. Winokur, G. K. Lum, M. R. Shaneyfelt, F. W. Sexton, G. L. Hash and L. Scott, "Use of COTS Microelectronics in Radiation Environments," in IEEE Trans. on Nuclear Science, Vol. 46, Issue 6, December 1999.
- [54] Anthony Jordan, "RadHard-By-Design integrated circuit suppliers: A success story," (website) <http://www.mil-embedded.com/articles/authors/jordan/>, November 2006.
- [55] Gerald W. Caldwell. "Implementation Of Configurable Fault Tolerant Processor (Cftp) Experiments." Master's Thesis, Naval Postgraduate School, Monterey, California, December 2006.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Professor Hersch Loomis
Naval Postgraduate School
Monterey, California
4. Professor Alan Ross
Naval Postgraduate School
Monterey, California
5. Gaspar M. Perez Casanova
Naval Postgraduate School
Monterey, California